

a row of fixed slots, and the one object that never lets you edit its own characters



## Declaring and filling a 1D array \*

- `int[] marks = new int[5];` creates an array of 5 int slots, each holding the default value 0 until it is set
- an array literal, written as `int[] marks = {9, 3, 7, 1, 5};` declares and fills the array in one line, sizing it from how many values are listed
- every slot is reached by an index, and Java indexes start at 0, so the first mark is `marks[0]`, holding 9, and the last is `marks[4]`, holding 5
- a for loop from 0 up to but not including `marks.length` visits every slot exactly once, whatever the array's actual size turns out to be



index 2 holds the value 7, counting starts at 0 so this is the third slot

## Arrays have exactly one built-in property: length

- `marks.length` gives the number of slots in the array, with no parentheses, since length is a property here and not a method
- `marks.length` never changes once the array is created, an array in Java is a fixed size from the moment it is made
- the highest legal index is always `marks.length - 1`, and using `marks.length` itself as an index fails at run time
- a 2D array such as `int[][] grid = new int[3][4];` is really an array of arrays, 3 rows each holding 4 int slots, reached as `grid[row][col]`

## array.length against String.length()

### Array

- length is a field, written without parentheses: `marks.length`
- fixed at creation, an array can never grow or shrink

### String

- `length()` is a method, written with parentheses: `word.length()`
- a String object never changes either, but a new String can be built from parts of it

writing `marks.length()` or `word.length` is a compile error, this exact pairing is the classic slip

