* Array

int[] grades;

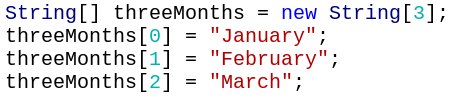
double[] febTemps;

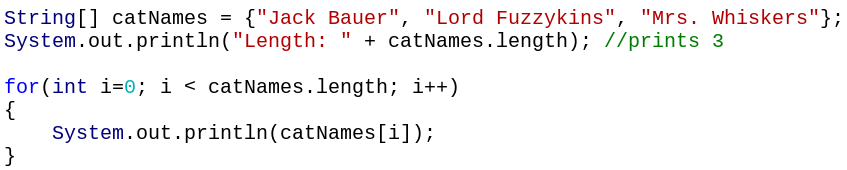
String[] args;

Note the square brackets, which means this variable is an array that stores elements of that type.

Arrays are created with the **new** keyword.

The length of an array can’t be changed after it’s created.





* we can’t use the braces to assign values to an array after it is created

Accessing arrays:

* Change element in the array:

grades[1] = 100;

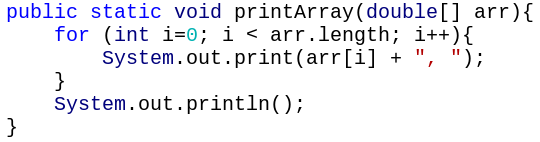
* Read value from the array:

int x = grades[5];

* Combination:

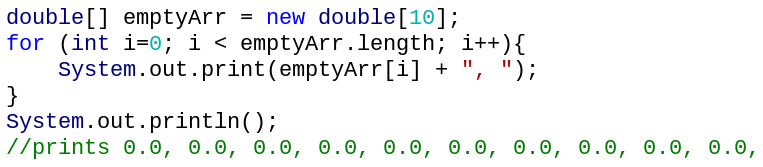
grades[3] = grades[5];

Printing arrays:



For Strings, the length is s.length()

For arrays, the length is catNames.length

Default values

For int/double arrays: 0

For boolean arrays: false

For char arrays: a special value

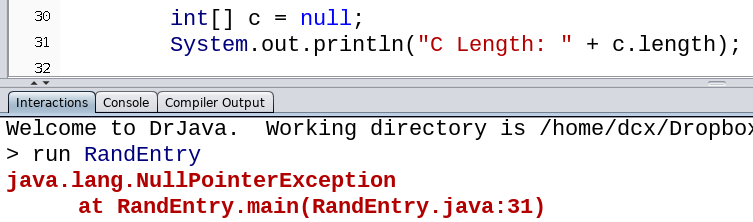
For String arrays: null

* Null

Reference type variables can store the **null** value.

Null means no address. (The website address is a big red X)

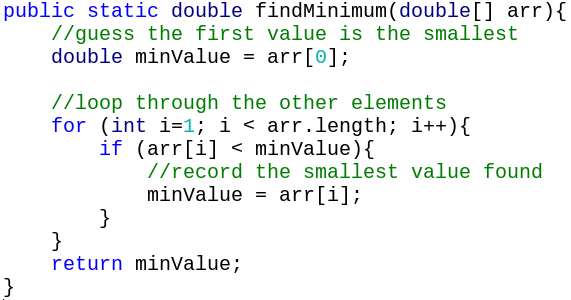
Null is useful to check if something has not been initialized yet.



**run-time error**

Occurs when a reference variable has the value null and you try to access it

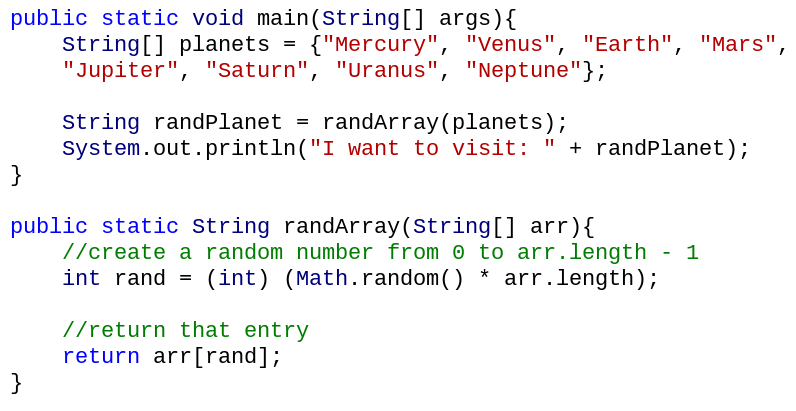
* Ex: Find Minimum



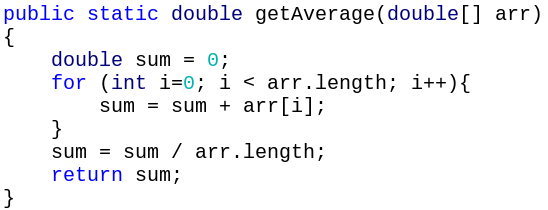
Note that this method will crash if the array is empty (arr[0])

With an **ArrayIndexOutOfBoundsException**

* Ex: Get random entry

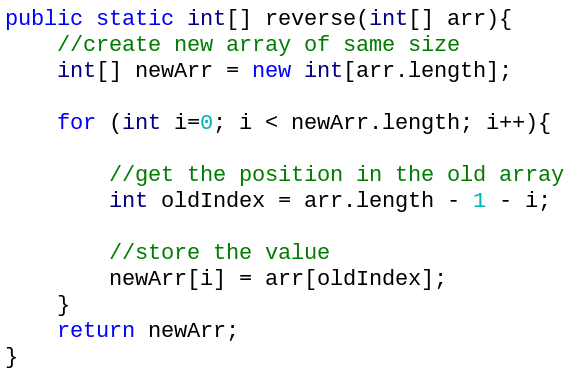


* Ex: Find the average of the array

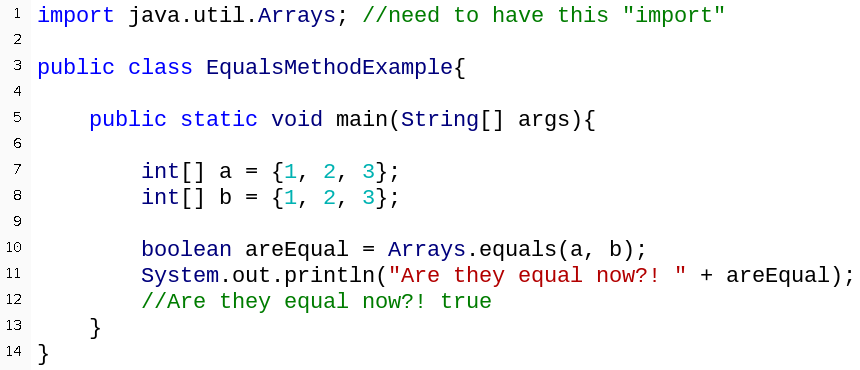


Be on the lookout for **integer division** here!

* Ex: Reverse array

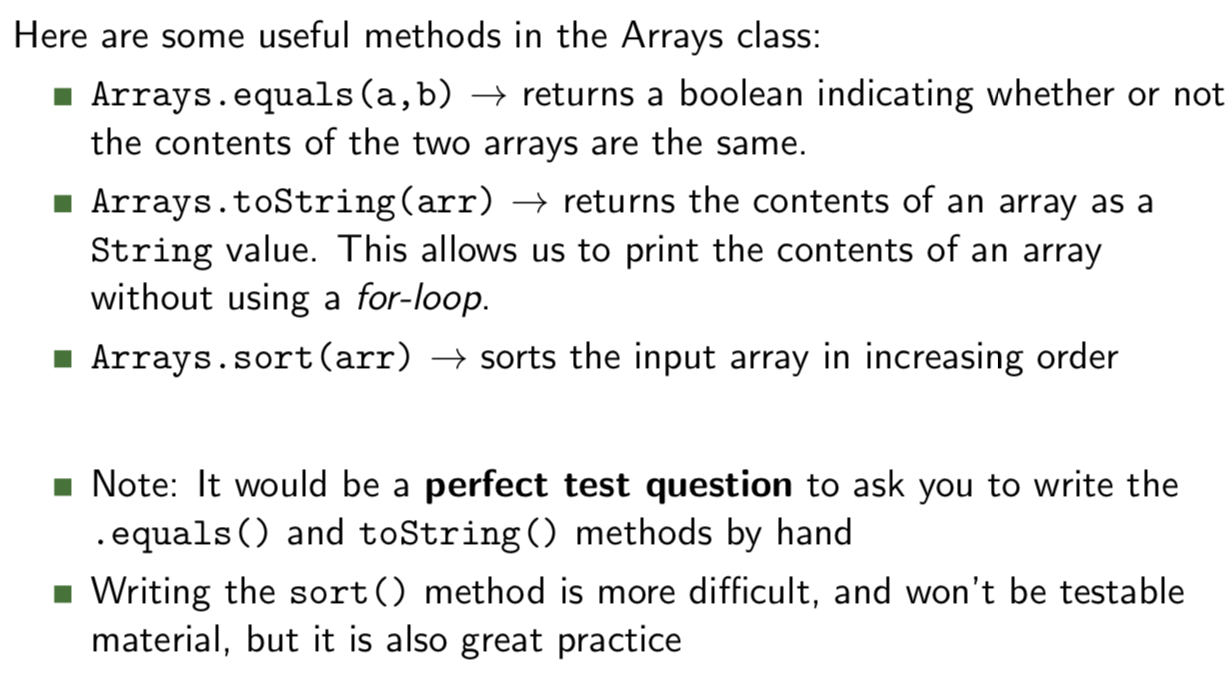


* built-in array methods



The top line is an example of an **import statement.**

All your import statements go at the top of java file, **before public class**.



test questions: write .equal() method and .toString() method

.sort() not testable

* Primitive vs. reference type

Primitive types: int, double, boolean, char

Reference types: String, Arrays, Objects

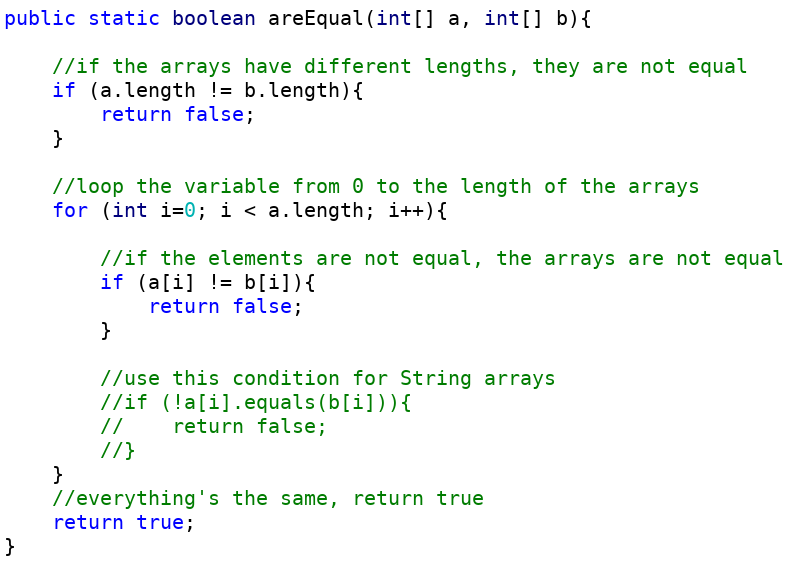
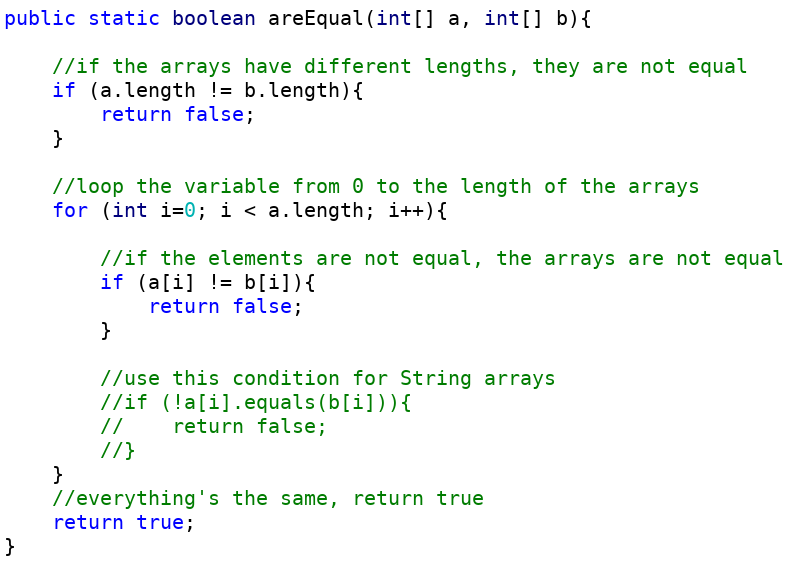
Reference type variables store **addresses** instead of values

* printing out their value might not work
* have to compare them a different way

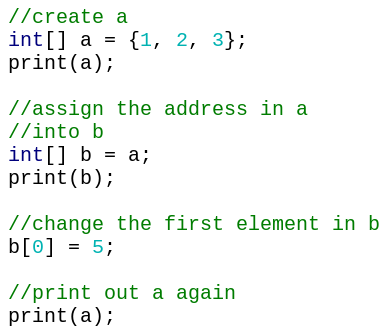
primitive types: ==

Strings: .equals()

* == is comparing the addresses



* can have two variables storing the same address (**aliasing**)



5 2 3

1 2 3

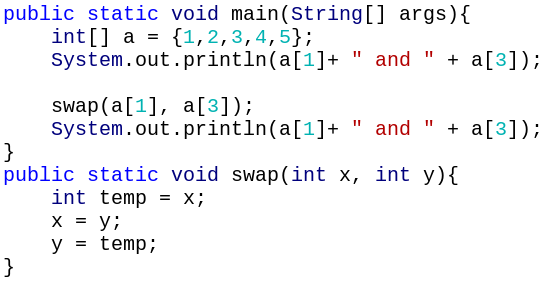
1 2 3

They both contain the same website address, so changes are seen for both.

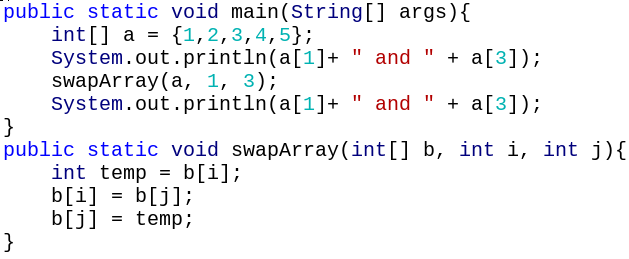
* Swapping

We are passing two int **values**, which are copied to the parameters of swap

Only the **variables** in swap are being swapped



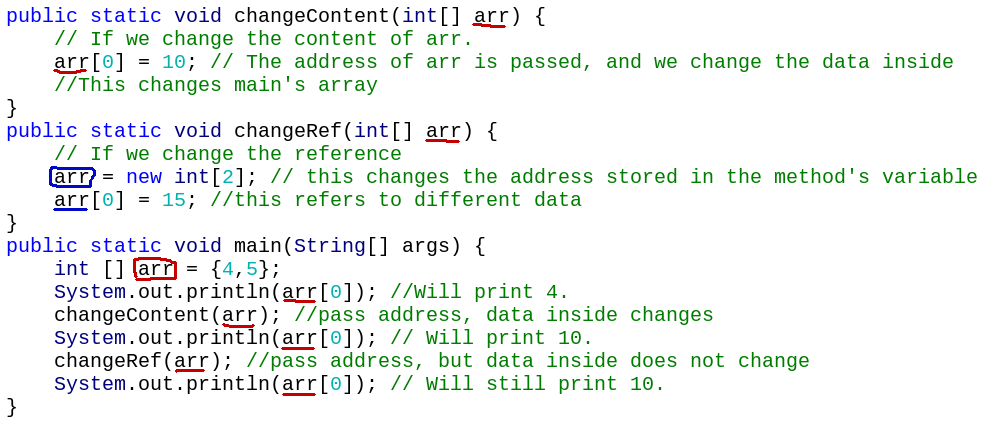
2 4



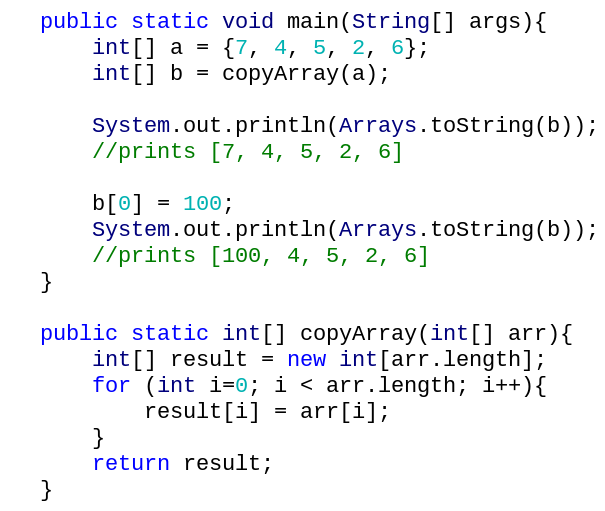
You must **pass an array** to a method if you want to change values within the array inside the method.

4 2

* create a new array



* copy an array



different addresses

* creating a 2D array
* **int [][] a = {{1, 2, 3}, {5, 6}}**
* **int [][] b = new int [2] [3]**

Because the array store integers, every entry will start off with a value of 0.

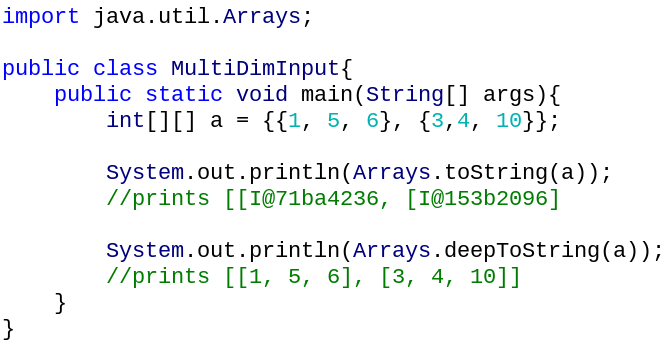
The array will contain [ [0, 0, 0], [0, 0, 0] ]

* **int [][] c = new int [2] []**

Each element in the array will be an array. However, what is currently stored is the value null

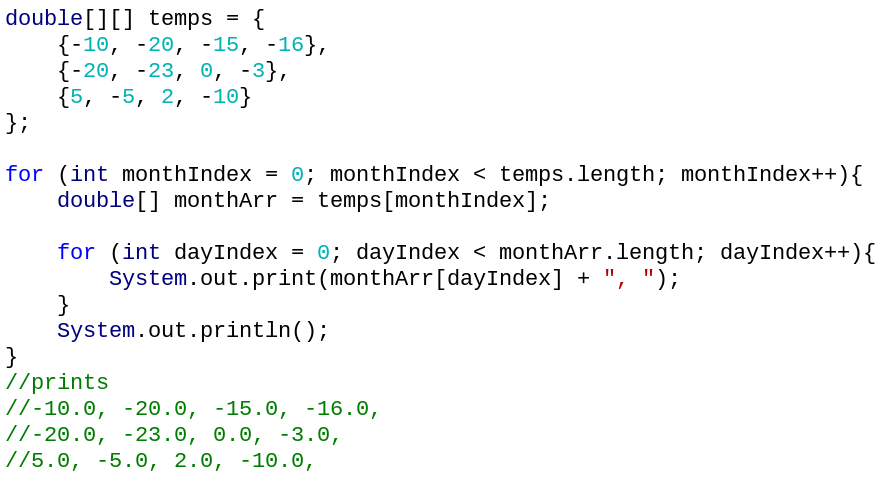
------Represents an uncreated array.

The array will contain [ null, null ]

* print 2D arrays

**Arrays.deepToString(arr)**

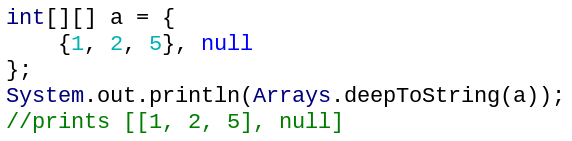
prints all of the elements of arr, where arr is a multi-dimensional array

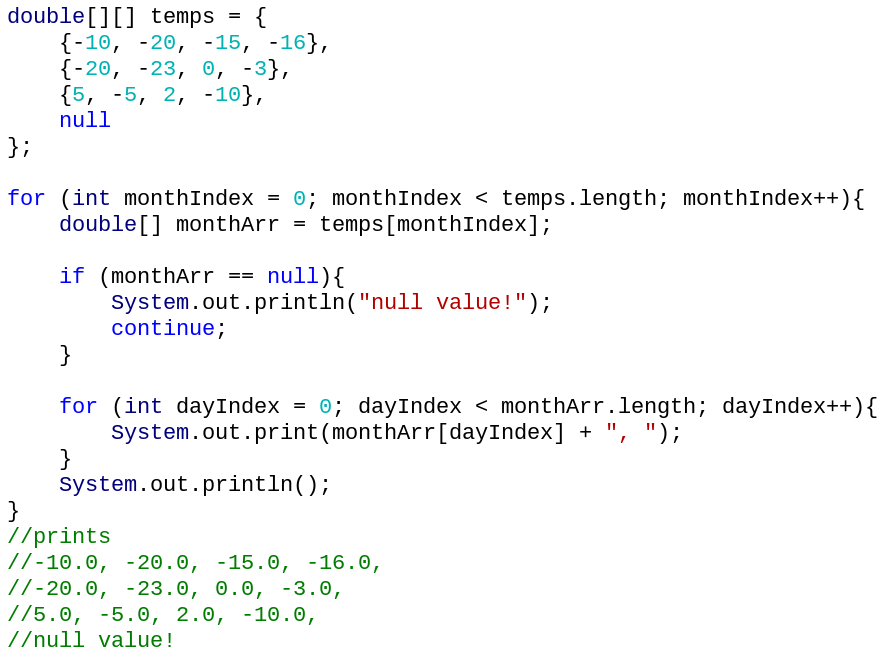


* Null in Arrays

Note that it’s possible to have null inside an array of arrays.

This represents an uninitialized element.



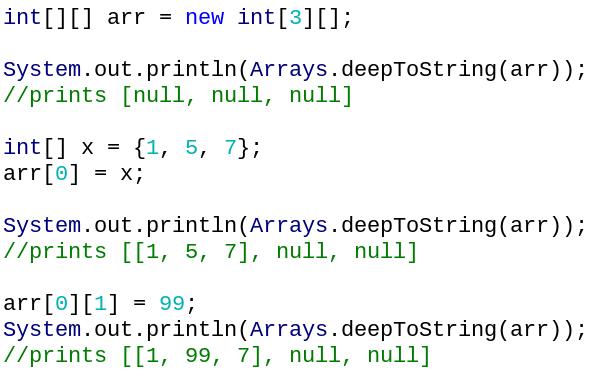


if not test,

may crash

It’s okay to use == here because we are testing to see if the address is null

* Input elements in the arrays

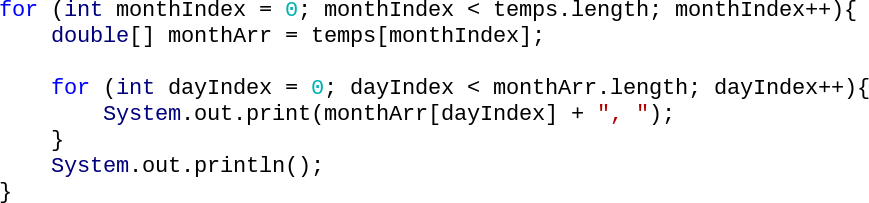


Can index into the outer and the inner array at once to change elements.

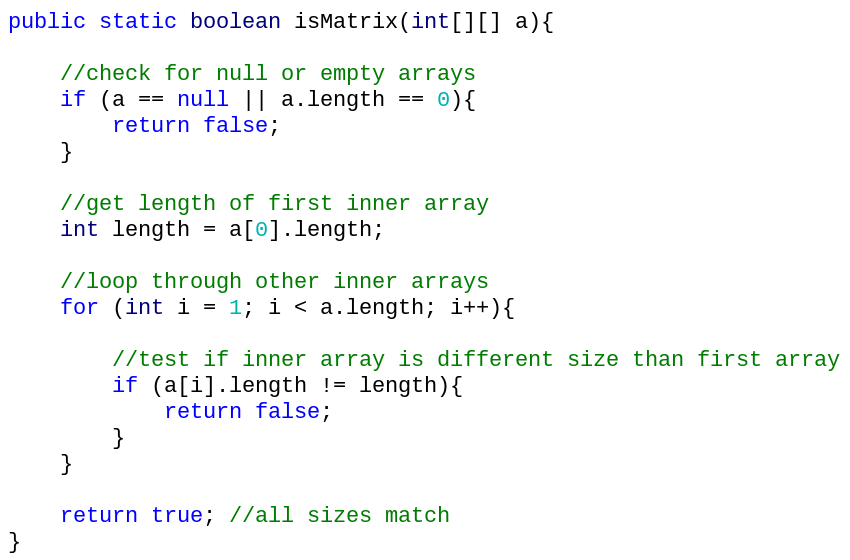
* Jagged Arrays

Sometimes, we may have arrays of arrays of **different lengths**.

Be careful on your for-loop that you don’t make assumptions about the inner array’s length.

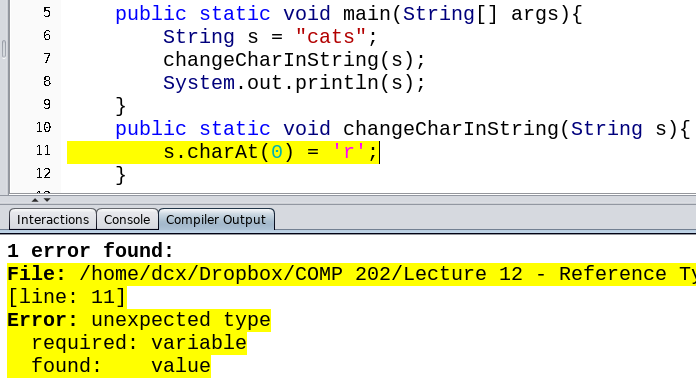


* IsMatrix

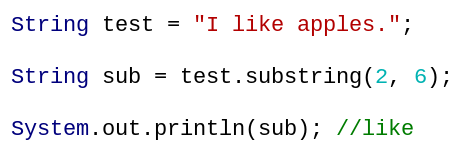


* String

Strings are immutable, which means the data can’t be changed



s.charAt(0) returns a char ’c’

Java can’t assign a new value to ’c’

* substring

public String substring(int start, int finish):

Returns a new String composed of the this String starting from index start and up to, but not including index of finish.

