Fundamentals of Programming 2

Lecture 6

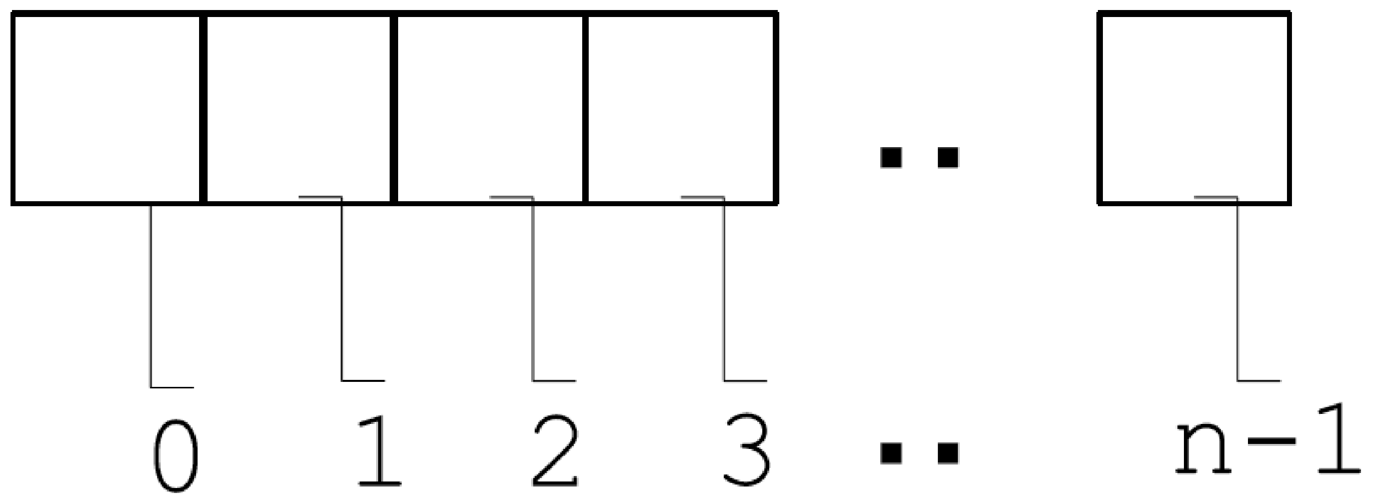
Aurelia Power, TU-Dublin, Blanchardstown Campus, 2019

* Notes based on the Java Oracle Tutorials (2019), Deitel & Deitel (2015), and Horstman (2013)

RECAP… Arrays

- An array is an indexed list of values.
- You can make an array of any type:

  int, double, String, etc..
- All elements of an array must have the same type
- Arrays define contiguous spaces in memory



RECAP… Arrays

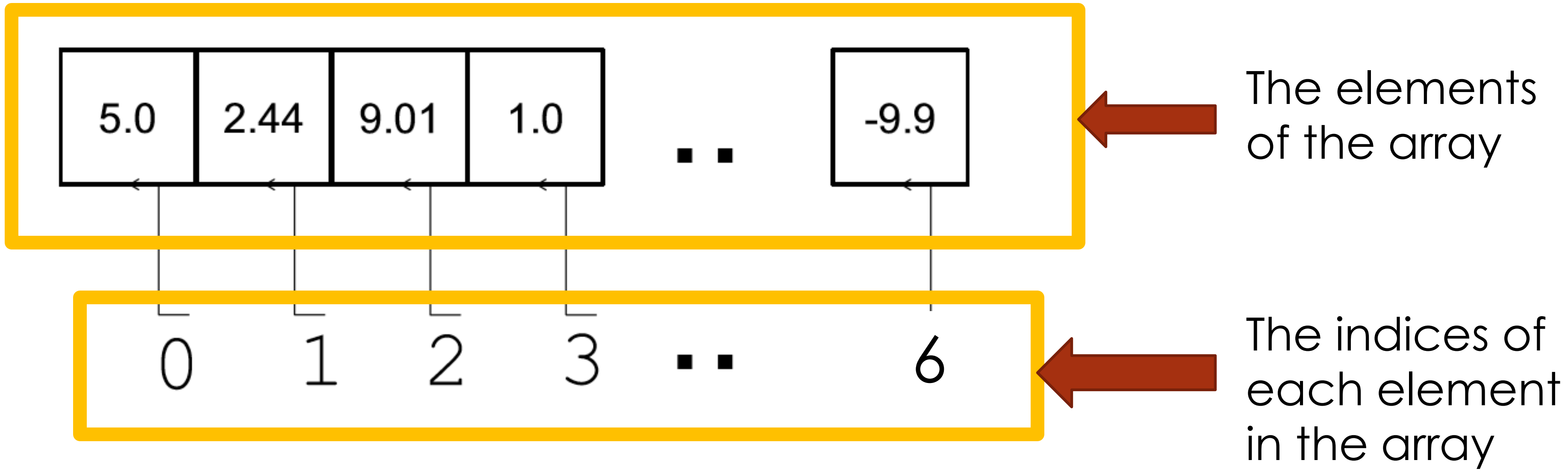
- Example:

```
double [ ] values = {5.0, 2.44, 9.01, 1.0, 7.3, 2.0, -9.9};
```

The size/number of elements is **7**

The first index is **0**;

The last index is **values.length – 1** (which is **6**);


5.0
2.44
9.01
1.0
..
-9.9
0
1
2
3
..
6
The elements of the array
The indices of each element in the array

RECAP… Arrays

- The index starts at **zero** and ends at **length-1**.

- Another Example:

```
int[ ] values = new int[5];
values[0] = 12; // CORRECT
values[4] = 12; // CORRECT
values[5] = 12; /* WRONG!! compiles but throws an
                   Exception at run-time */
```

RECAP… Arrays

- An array is declared using: DATATYPE[ ] name;
- Arrays are just another reference data type:

```
int[ ] values1; // array of int … unidimensional
int[ ][ ] values2; /* array of arrays of int… two-
                      dimensional */
```

- Arrays are **NOT** primitive data types
- Arrays are data structures that store a collection of **same type values** (primitive or object references) **contiguously in memory.**

```
double[ ] temperatures; // array of primitive data type
String[ ] names; // array of String object references
```

RECAP… Arrays

- To create an array of a given size, use the operator new

```
int[ ] values = new int[5]; /*creates an array of 5 ints, all
                                     initialised to 0 */
```

- or you may use a variable to specify the size:

```
int size = 12;
int[ ] values = new int[size]; /*creates an array of 12 ints, all
                                        initialised to 0 */
```

- **REMEMBER:** size must always be an int value, whether a specific value, or a variable, or an expression that resolves to an int.

RECAP… Arrays

- Curly braces can be used to **initialize** an array; it can ONLY be used when you declare the variable.

```
int[ ] values  = { 12, 24, -23, 47 };
```

- To **acces**s the elements of an array, use the [ ] operator: values[index]

Example:

```
int[ ] values  = { 12, 24, -23, 47 };
values[3] = 18;        // {12, 24, -23,18} … So it replaced 47 with 18
int x = values[1] + 3;  /* It takes the value of the second element of the
                        array which is 24, adds 3 to it, and assigns that to the
                        variable x…so x has the value of 27. Note that x is
                        not an element of the array…just another variable
                        */
```

RECAP… Arrays

- Each array has a ***length* variable built-in** that stores the number of elements the array it has:

```
int[ ] values = new int[12];
int size =  values.length; // 12
int[ ] values2 = {1,2,3,4,5};
int size2 =  values2.length; // 5
```

- Once created, the size/length of the array CANNOT be changed

RECAP… Loops and Arrays

- Can initialise arrays to custom values using loops:

```
int[ ] values = new int[12];
for (int i = 0;  i <
 values.length; i++) {
     values[i] = i;
}
```

What values will the elements of the array will have??

0 1 2 3…11

- Can initialise arrays to custom values using loops:

```
int[ ] values = new int[12];
int i = 0;
while (i < values.length) {
     values[i] = i;
     i++;
}
```

What values will the elements of the array will have??

The same 0 1 2 3…11, but this time I used a while loop

RECAP... Loops and Arrays

- We can initialise the values of the array elements to user input by using a loop to continuously take user input:

```
String[] names = new String[10];
for(int i = 0; i < names.length; i ++){
        System.out.print ("Enter a name: ");
        names[i] = sc.next();
}
```

- Then we can examine and process its elements, again, by using a loop structure...

```
//go through each element of the array (each name)
for(int i = 0; i < names.length; i ++){
        //if its length is 5 or more print it
        if(names[i].length() >= 5){
                System.out.print(names[i] + " ");
        }
        //if it's not we do nothing… so we don't need an else
}
```

It prints all the names that have 5 or more characters...

and that's (more or less) exercise 1 from lab 5 ... you need to write the import statements, declare the class and the main method, compile and run.

RECAP… Loops and Arrays

- What about this loop???

```
int[] arr = new int[10];
for(int i = 0; i < arr.length; i++){
        arr[i] = (int)(Math.random()*50)+1;
}
```

It initialises each element of the array to random values between 1 and 50 both inclusive.

- What about this loop?

```
for(int i = 0; i < arr.length; i+=2){
        System.out.print(arr[i] + " ");
}
```

It prints all the elements at even indices… NOT the even elements!!

- What about this loop?

```
for(int i = 0; i < arr.length; i++){
        if(arr[i] % 2 != 0){
                System.out.print(arr[i] + " ");
        }
}
```

It prints all the odd elements…

and that's (more or less) exercise 4 from lab 5 … you need to declare the class and the main method, compile and run.

RECAP… Loops and Arrays

- We can even use enhanced loops with arrays because they are iterable structures:

```
int product = 1;
for(int element : arr){
    product *= element;
}
System.out.println(product);
```

It calculates the product of all the elements in the array …

```
int negCount = 0;
for(int element : arr){
    if(element < 0){
        ++negCount;
    }
}
System.out.println(negCount);
```

It counts the number of negative elements in the array …

and that's (more or less) exercise 3 from lab 5 … you need to write the declare the class, the main method, declare and initialise arr, compile and run.

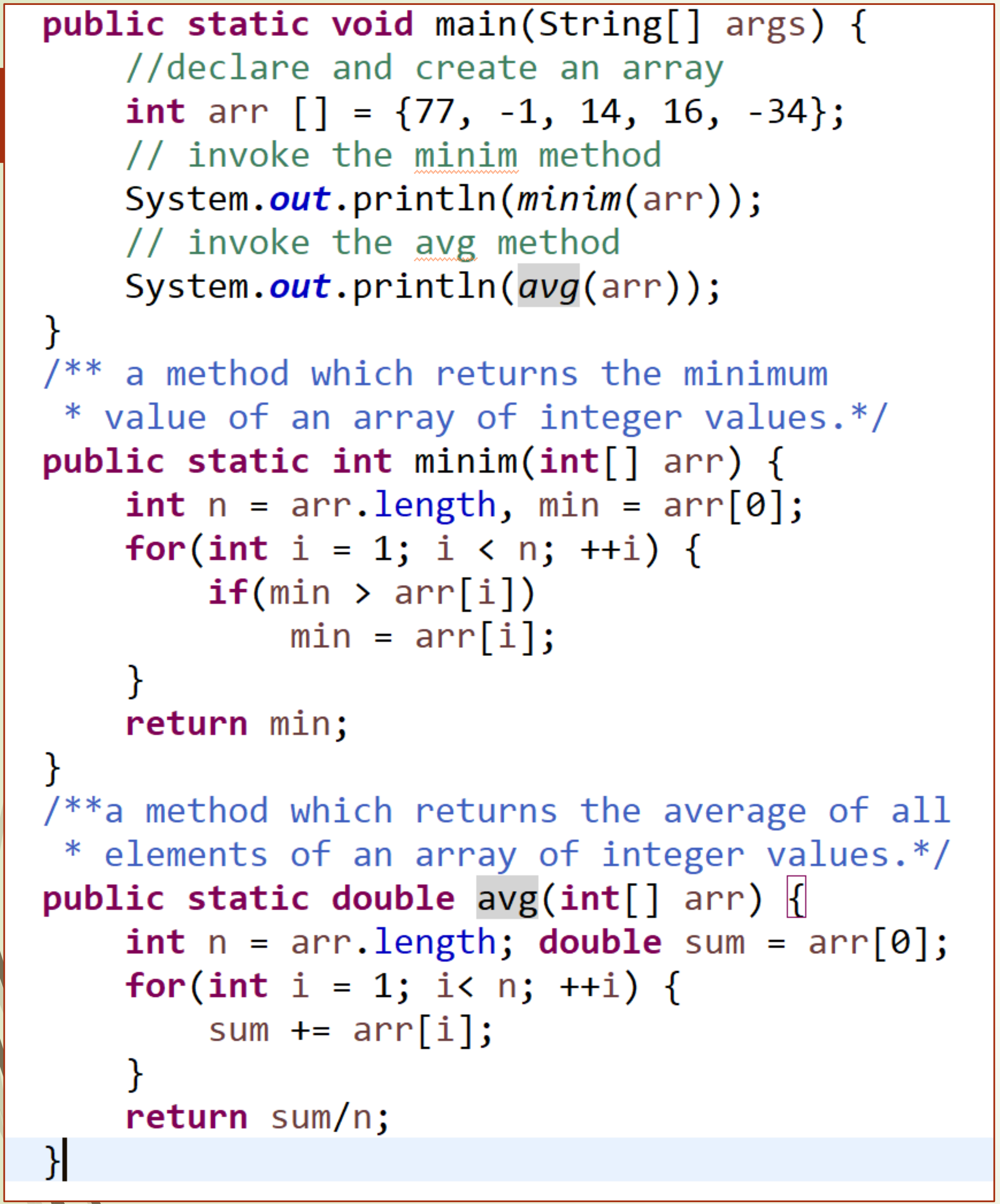
RECAP… Methods and Arrays

- Arrays can be used with methods:
  - They can be passed to a method
  - They can also be returned from a method

```
public static int[] someSuitableMethodName (int[] values){
    int[] newValues = new int[values.length];
    for(int i = 0; i < values.length; i++) {
        newValues[i] = values[i] + 1;
    }
    return newValues;
}
```

It returns a new array that contains the original values to which 1 is added

Lab 5 – Exercise 2



```java
public static void main(String[] args) {
    //declare and create an array
    int arr [] = {77, -1, 14, 16, -34};
    // invoke the minim method
    System.out.println(minim(arr));
    // invoke the avg method
    System.out.println(avg(arr));
}
/** a method which returns the minimum
 * value of an array of integer values.*/
public static int minim(int[] arr) {
    int n = arr.length, min = arr[0];
    for(int i = 1; i < n; ++i) {
        if(min > arr[i])
            min = arr[i];
    }
    return min;
}
/**a method which returns the average of all
 * elements of an array of integer values.*/
public static double avg(int[] arr) {
    int n = arr.length; double sum = arr[0];
    for(int i = 1; i< n; ++i) {
        sum += arr[i];
    }
    return sum/n;
}
```

- You need to declare the class, then compile and run it.

RECAP … Two-dimensional arrays

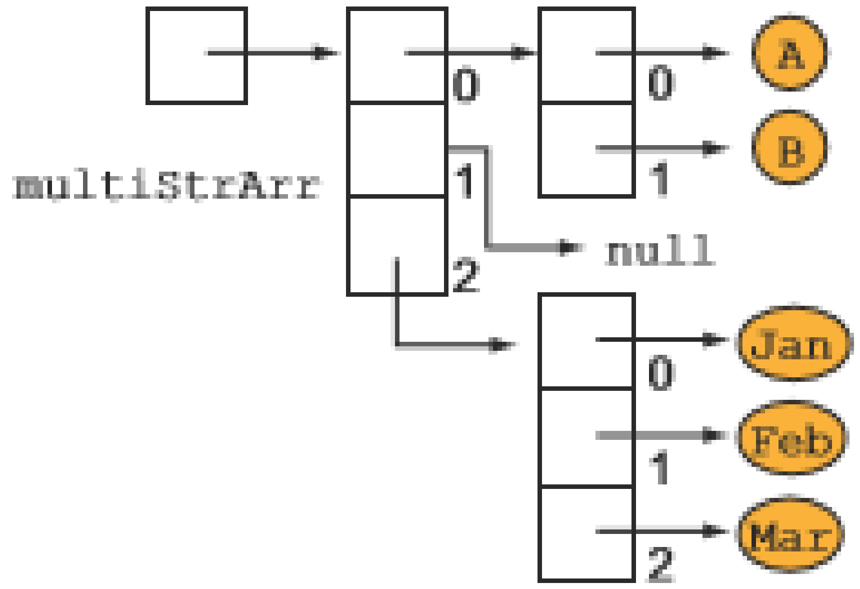
- Typically used to store tabular data;
- You create a 2D array by specifying the number of rows and columns;
- A 2D array is an array of 1D arrays, all the same data type.
- To access a particular individual element you need to specify the name of the array and both indices, corresponding to the row and column, respectively:

```
int n  = data[0][1];  //which is 3
```

indices	Column 0	Column 1	Column 2	Column 3
Row 0	16	3	2	13
Row 1	5	10	11	8
Row 2	9	6	7	12
Row 3	4	15	14	1

RECAP … Asymmetrical 2D arrays

```
String multiStrArr[][] = {
                {“A”, “B”},
                null,
                {“Jan”, “Feb”, “Mar”}
        };
```



- Arrays of object references can point to the value ***null;***
- but we must be careful, because we might get a NullPointerException;
- For instance, trying to access the element **multiStrArr[1][0]** will throw an exception…because the component array **multiStrArr[1]** is null and does not contain itself any Strings.

RECAP… Loops and 2 D Arrays

- 2D arrays are typically processed using nested loops:

```
    String[ ][ ] names = {
names[0][0]      {‘Helen’, “Adam”, “Ray”, },            names[0]
                 {“Jane”, “John”},            names[1]
                 {“Luca”, “Arthur”, “Aurelia”, “Gerry”}            names[2]
       };          names[2][1]
    for (int row = 0; row < names.length; row++){
       for (int column = names[row].length -1; column >= 0; column--){
          System.out.print(names[row][column] +  “  ”);
       }
       System.out.println();
    }
```

Each row of names will be printed on a different line; the names in each row will be in reverse order… exercise 5 from lab 5 (need to declare the class, the main, compile and run)

Today

Sorting arrays:

- Bubble sort
- Selection sort

Searching arrays:

- Linear search

Sorting

- One of the most common tasks in data processing

- Examples that entail sorting:

✓ Students displayed in alphabetical order

✓ Numbers displayed in ascending/increasing order

✓ Numbers displayed in descending/decreasing order

Sorting a collection of data is the task of re-arranging that data in a particular order.

Sorting

- There are many **sorting algorithms**:
    - **Selection sort**
    - **Bubble sort**
    - Insertion sort
    - Merge sort
    - Etc.
- **What is an algorithm?**

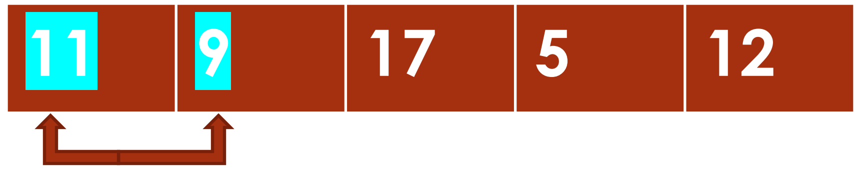
a set of steps/instructions to solve a specific problem (almost like a recipe for cooking)

Ascending Bubble Sort Algorithm

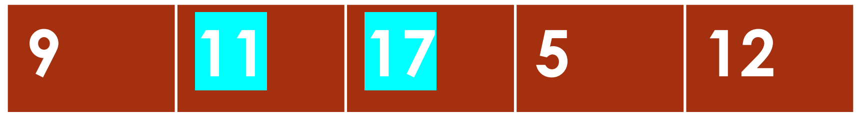
- Bubble sort works by comparing each **adjacent pair of items** in a list, swapping the items if the first element of the pair is greater than the second element of the pair (or smaller in the case of descending sort),
- It repeats the above through the entire list until no more swaps can be done.
- It can work from either the first item moving larger items towards the back of the list, or from the back moving smaller items towards the front.
- It is sometimes called sinking sort or exchange sort.
- It can be slow with large arrays.
- Performance can be improved by modifying it to a bi-directional bubble sort, i.e. one pass from front to back, the next from back to front (but we will focus only on the uni-directional sort … more intuitive)
- Can be implemented for **descending order and for any data types**

Ascending Bubble Sort Algorithm

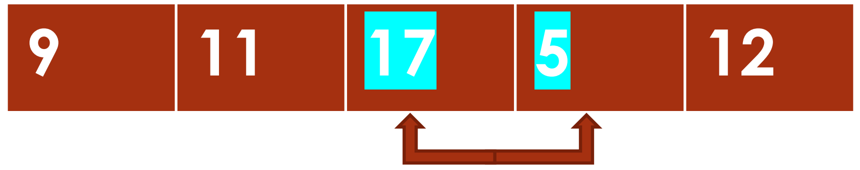
EXAMPLE: `int [ ] array = {11, 9, 17, 5, 12};`



1. Look at the first pair of elements: array[0] (which is 11)and array[1] (which is 9) and swap them because the first one is greater than the second one: so array[0] will be 9 and array[1] will be 11

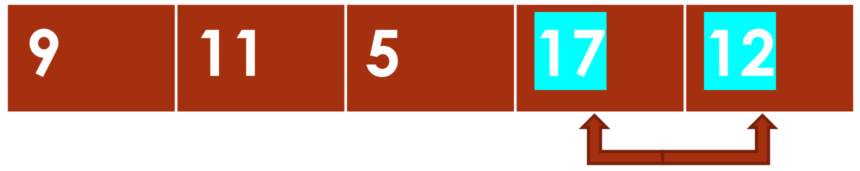


2. Look at the next pair of elements: array[1] (which is 11)and array[2] (which is 17) and do nothing because the first one is smaller than the second one.. No need to swap them.

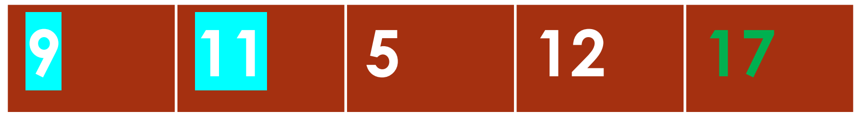


3. Look at the next pair of elements: array[2] (which is 17)and array[3] (which is 5) and swap them because the first one is greater than the second one: so array[2] will be 9 and array[3] will be17

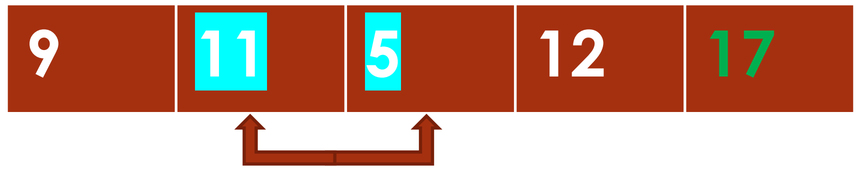
Ascending Bubble Sort Algorithm



4. Look at the next pair of elements: array[3] (which is 17)and array[4] (which is 12) and swap them because the first one is greater than the second one: so array[3] will be 12 and array[4] will be 17; therefore the last element will be in place/ordered since it is the largest/greatest

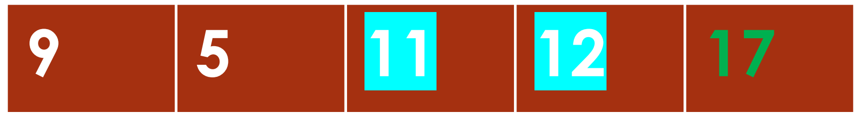


5. Start again and look at the first pair of elements: array[0] (which is 9)and array[1] (which is 11) and do nothing because the first one is smaller than the second one.

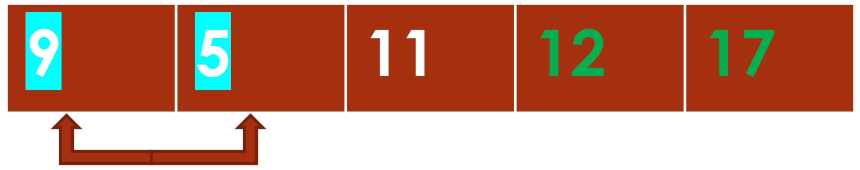


6. Look at the next pair of elements : array[1] (which is 11)and array[2] (which is 5) and swap them because the first one is greater than the second one: so array[1] will be 5 and array[2] will be 11

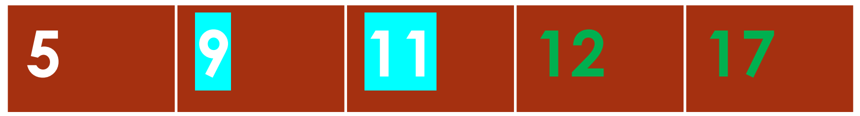
Ascending Bubble Sort Algorithm



7. Look at the next pair of elements: array[2] (which is 11)and array[3] (which is 12) and do nothing because the first one is smaller than the second one; now, the element at array[3] will be in place/ordered since it is the second largest/greatest element in the array



8. Start again and look at the first pair of elements: array[0] (which is 9)and array[1] (which is 5) and swap them because the first one is greater than the second one: so array[0] will be 5 and array[1] will be 9



9. Look at the next pair of elements : array[1] (which is 9)and array[2] (which is 11) and do nothing because the first one is smaller than the second one; now, the element at array[2] will be in place/ordered since it is the third largest/greatest element in the array

Ascending Bubble Sort Algorithm

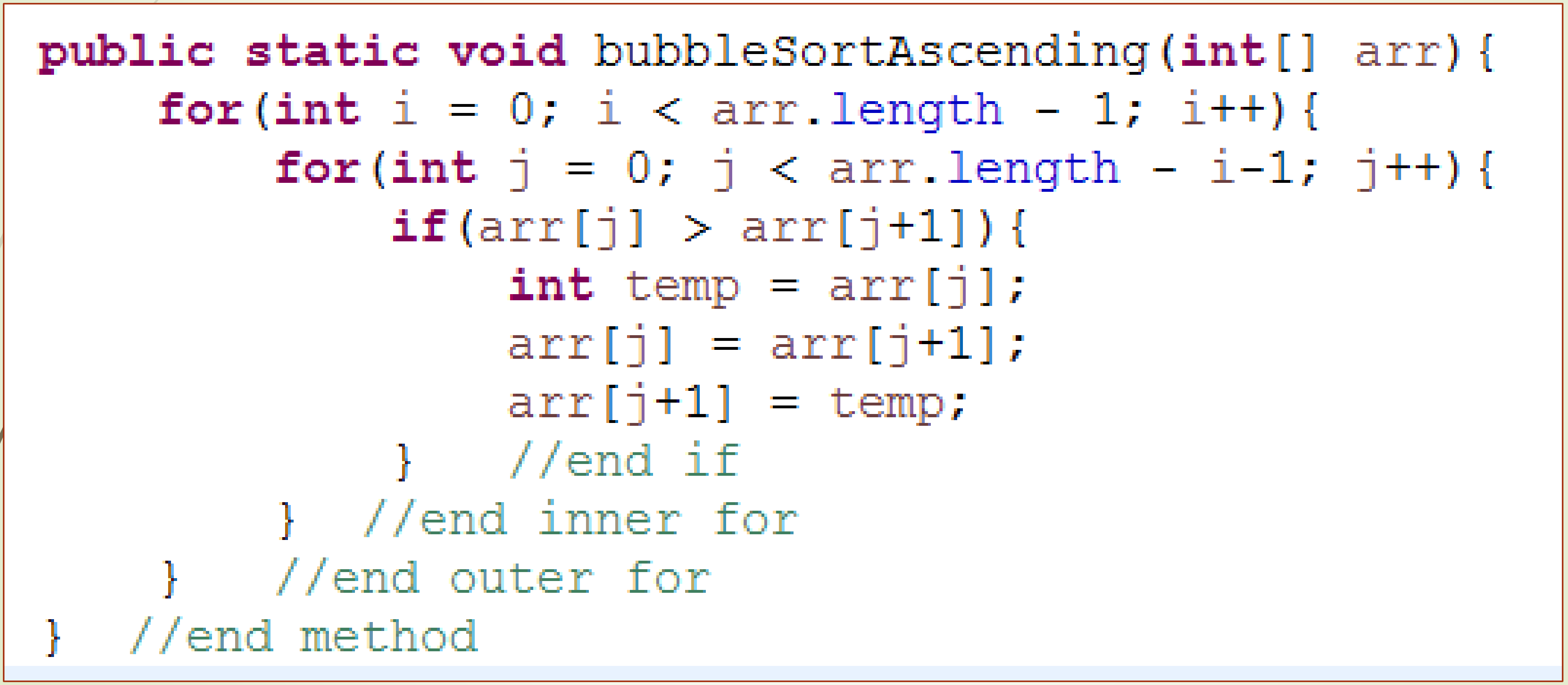
5	9	11	12	17

10. Start again and look at the first pair of elements: array[0] (which is 5)and array[1] (which is 9) and do nothing because the first one is smaller than the second one; now, the element at array[1] will be in place/ordered since it is the fourth largest/greatest element in the array; but so is the element at array[0] since there is no other element to compare it with.

5	9	11	12	17

11. Now the array is sorted.

Ascending Bubble Sort Algorithm – java implementation



```
public static void bubbleSortAscending(int[] arr){
    for(int i = 0; i < arr.length - 1; i++){
        for(int j = 0; j < arr.length - i-1; j++){
            if(arr[j] > arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }   //end if
        }  //end inner for
    }   //end outer for
}  //end method
```

Ascending Bubble Sort Algorithm - exercise

- How does the bubble sort algorithm modifies the following sequence **5 4 3 2 1** to sort it in ascending order ?

5 4 3 2 1 → 4 5 3 2 1 → 4 3 5 2 1→ 4 3 2 5 1 → 4 3 2 1 5 → 3 4 2 1 5 → 3 2 4 1 5 → 3 2 1 4 5 → 2 3 1 4 5 → 2 1 3 4 5 → 1 2 3 4 5

- Why do we need another **variable temp**? Could we not simply swap as follows: arr[j] = arr[j+1]; arr[j+1] = arr[j]; ?

Because the old value of arr[j] will be wiped out/lost and replaced with the value of arr[j+1], ending up with both variables having the same value of arr[j+1]; so we need a temporary variable to store the old value of arr[j].

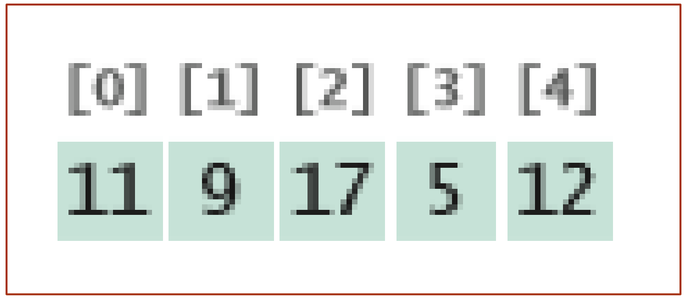
Ascending Selection Sort Algorithm

- Is a simple but also inefficient algorithm for large amounts of data;
- It is based on the idea of finding the index of the **smallest remaining element** of the array, given k^{th} iteration: pick the smallest element and swap it with the first one; pick the smallest element of the remaining ones and swap it with the next one, and so on.
- The algorithm summarised:
  1. Find the smallest element of the array and swap it with the element in the first position of the array
  2. Find the next smallest element in the array and swap it with the element in the second position of the array
  3. Continue until the array is sorted
- Can be implemented for descending order and for other data types

Ascending Selection Sort Algorithm

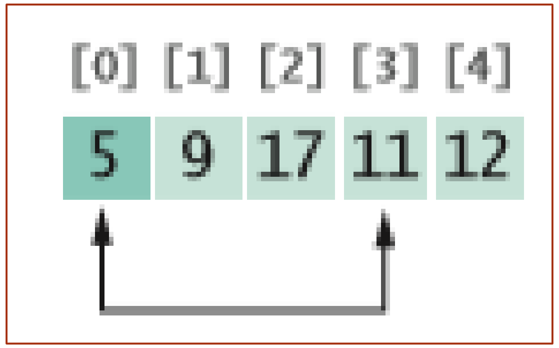
- This algorithm sorts an array by **repeatedly finding the smallest element of the unsorted tail region and moving it to the front.**

EXAMPLE: int [ ] array = {11, 9, 17, 5, 12};



1. Initial array: unsorted

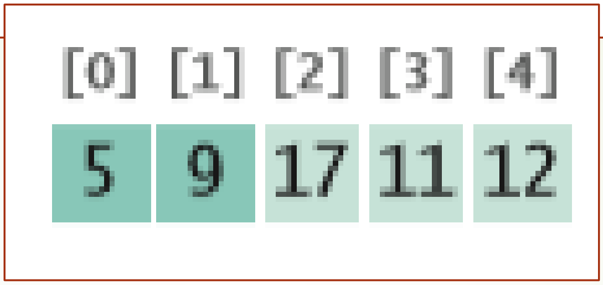
2. Find the smallest element which is 5 (the element at index 3) and swap it with the first element at index 0 (in this case 11)



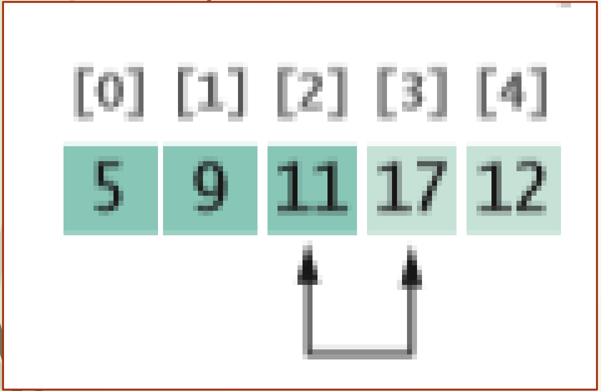
3. Now the first element is in the correct place, **already sorted**… so we no longer look at it

Ascending Selection Sort Algorithm

4. Next, we look at the remaining entries array[1] to array[4] and find the smallest element among them, which is 9 (element at index 1); because it is already in its correct place, we don't need to do anything.

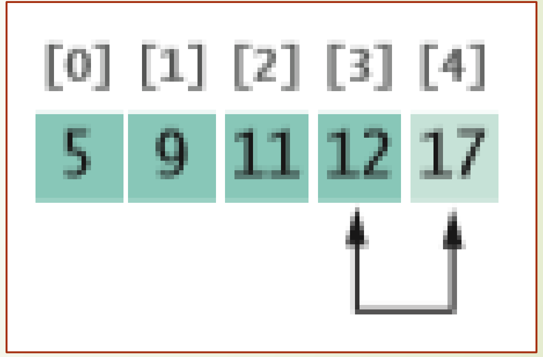


5. Next, we look at the remaining entries array[2] to array[4] and find the smallest element among them, which is 11 (element at index 3); because it's not in the correct place, we swap it with the element at index 2 (17).



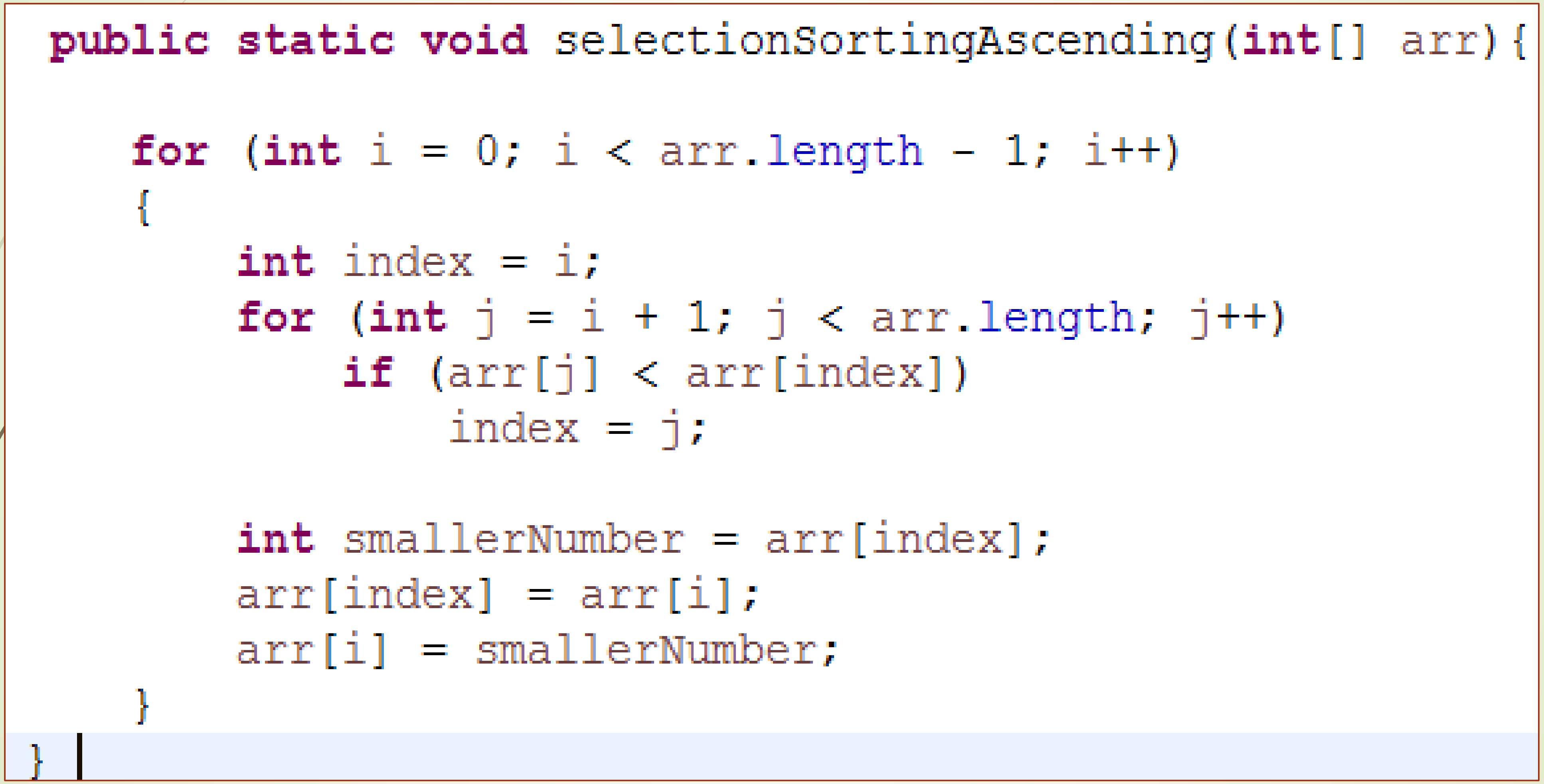
6. Now the first 3 elements are in the correct place, already sorted

7. Next, we look at the remaining entries array[3] to array[4] and find the smallest element among them, which is 12 (element at index 4); because it's not in the correct place, we swap it with the element at index 3 (17).



8. Now all the elements are in the correct place, sorted

Selection Sort Ascending – java implementation



```
public static void selectionSortingAscending(int[] arr){

   for (int i = 0; i < arr.length - 1; i++)
   {
       int index = i;
       for (int j = i + 1; j < arr.length; j++)
           if (arr[j] < arr[index])
               index = j;

       int smallerNumber = arr[index];
       arr[index] = arr[i];
       arr[i] = smallerNumber;
   }
}
```

Ascending Selection Sort Algorithm - quiz

- What steps does the selection sort algorithm go through to sort the following sequence in ascending order 6 5 4 3 2 1?

6 5 4 3 2 1 → 1 5 4 3 2 6→ 1 2 4 3 5 6 → 1 2 3 4 5 6

- Swap the values of the following 2 variables: **int a = 7; int b = 5;**

```
int temp = a;
a = b ;
b = temp;
```

Searching

- **Searching** is the task of looking for a particular value in a given data list/collection (such as an array);

- **Examples:**
  - Searching for a student name in the student directory;
  - Looking for a phone number in a phone book;
  - Searching for a book on Amazon
  - Searching for a name on Facebook to add to your list of friends.

Searching – Linear Search

- Goes through each element of the array, one at a time, and compares it with the **search key (the value you are searching for)**;
- if it finds a match, it will typically return the index in the array where the match was found;
- if it doesn't find any match after exhausting all the elements, it will return -1.

ALGORITHM:

```
For each element in the array:
        Compare the element at the given index with the search key:
                if they are the same, return the array index of that name
If all the names were looked at and no match was found, return -1;
```

QUESTION: What should happen to the loop when a match is found?

Answer: the loop should terminate/exit

Linear Search

Example:

Given an array of names, say John, Anna, Helen, Dermot, Brendan, search for the name Helen (the search key), and return the position where it occurs:

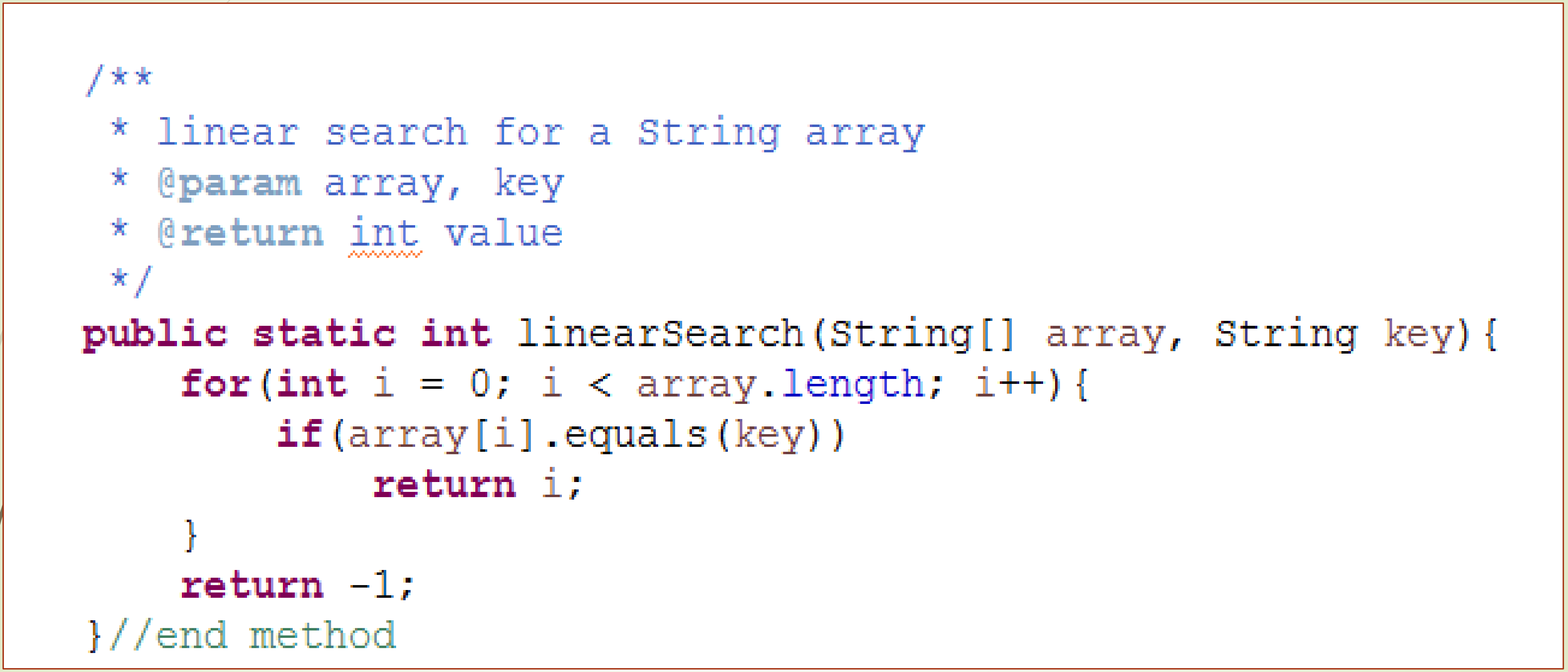
John	Anna	Helen	Dermot	Brendan
0	1	2	3	4
Helen	Helen	Helen		

Is the name at index 0 which is John the same as our search key Helen? No…look to the next name/element

Is the name at index 1 which is Anna the same as our search key Helen? No…look to the next name/element

Is the name at index 2 which is Helen the same as our search key Helen? Yes…so return 2 (the index of the name that matches the search key) and stop the search/loop

Linear Search - JAVA IMPLEMENTATION:



```java
/**
 * linear search for a String array
 * @param array, key
 * @return int value
 */
public static int linearSearch(String[] array, String key){
    for(int i = 0; i < array.length; i++){
        if(array[i].equals(key))
            return i;
    }
    return -1;
}//end method
```

Linear Search - Notes

- The array/collection does not need to be sorted;
- Suitable for relatively **small arrays**;
- For large arrays, it is **inefficient**;
- If there are more than one elements with the same value, it will return **the first occurrence**;

Your turn… True or false

- The linear search returns the element when a match is found…

False … it returns the index where a match was found, not the element.

- Linear search looks at all elements in the array.

False… once a match is found, the search is stopped; it only goes through all the elements when the match is the last element or there is no match at all.

- Linear search works by comparing the search key to each element of the array until a match is found, or all the elements are looked at.

True

- The search key must be an integer value.

False… the search key can be any data type as long as it is the same as what the array/collection stores.

- Using the method on slide 36, explain the steps it goes through when called with the following array {"Anna", "George", "John", "Elena", "Aristotle"} and the key "john".

It returns -1.