OBJECT ORIENTED PROGRAMMING UNIT4: STORAGE STRUCTURES

CFGS - DAW 1

JESUITAS

Sagrado Corazón
Jesuitas · Logroño

Index

4.1 Arrays

- An array is a special kind of object that holds zero or more primitive values or references.
- These values are held in the elements of the array, which are unnamed variables referred to by their position or index.
- The type of an array is characterized by its element type, and all elements of the array must be of that type.

4.1 Arrays

- Array elements are numbered starting with zero, and valid indexes range from zero to the number of elements minus one.
    - The array element with index 1, for example, is the second element in the array. The number of elements in an array is its length.
- <u>The length of an array is specified when the array is created, and it never changes.</u>
- All arrays have a public final int field named length that specifies the number of elements in the array.

4.1 Arrays

- Declare an int array variable. An array variable is a remote control to an array object:

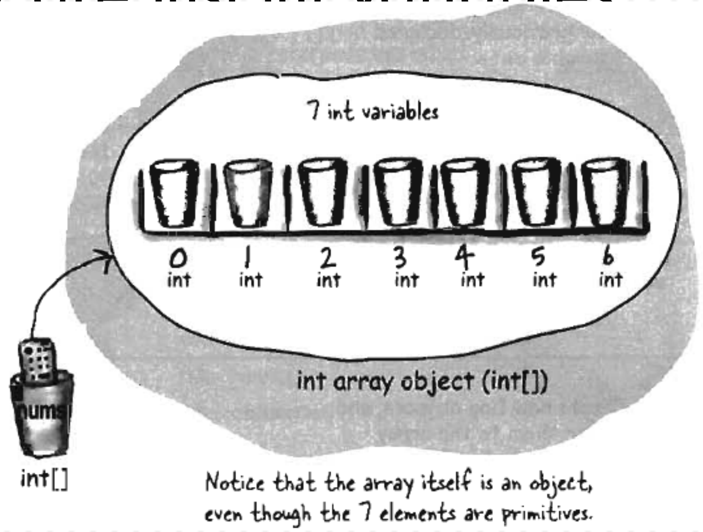
  int[] nums;

- Create a new int array with a length of 7 and assign it to the previouslyd eclared int variable *nums*:

  nums = new int[7] ;

- Give each element in the array an int value. Remember, elements in on int array are just int variables

  nums[0] = 6;
  nums[1] = 19;
  nums[2] = 44:
  nums[3] = 42:
  nums[4] = 10:
  nums[5] = 20;
  nums[6] = 21;

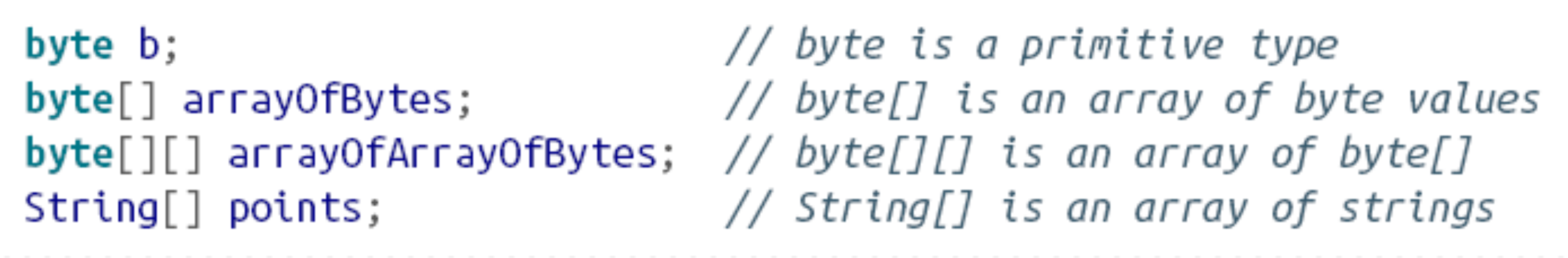

7 int variables
0 int
1 int
2 int
3 int
4 int
5 int
6 int
int array object (int[])
nums
int[]
Notice that the array itself is an object, even though the 7 elements are primitives.

4.2 Arrays are objects too

- Arrays are always objects, whether they're declared to primitives or object references. But you can have an array object that's declared to hold primitive values.
  - In other words, the array object can have elements which are primitives. But the array itself is never a primitive.
  - Array types are reference types, just as classes are.
  - Instances of arrays are objects, just as the instances of a class are.

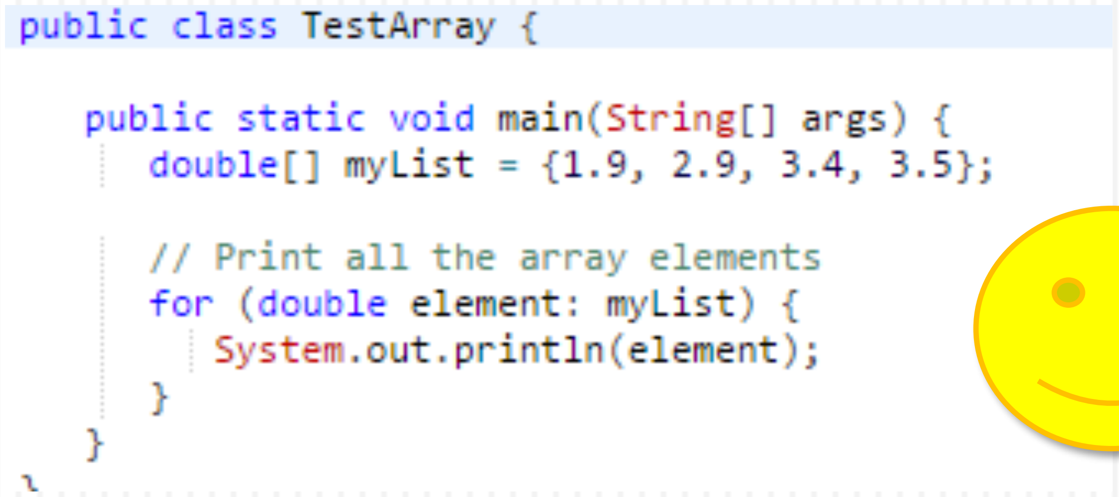
4.3 Array Types

- Unlike classes, array types do not have to be defined. Simply place square brackets after the element type. For example:



```
byte b;                     // byte is a primitive type
byte[] arrayOfBytes;        // byte[] is an array of byte values
byte[][] arrayOfArrayOfBytes; // byte[][] is an array of byte[]
String[] points;            // String[] is an array of strings
```

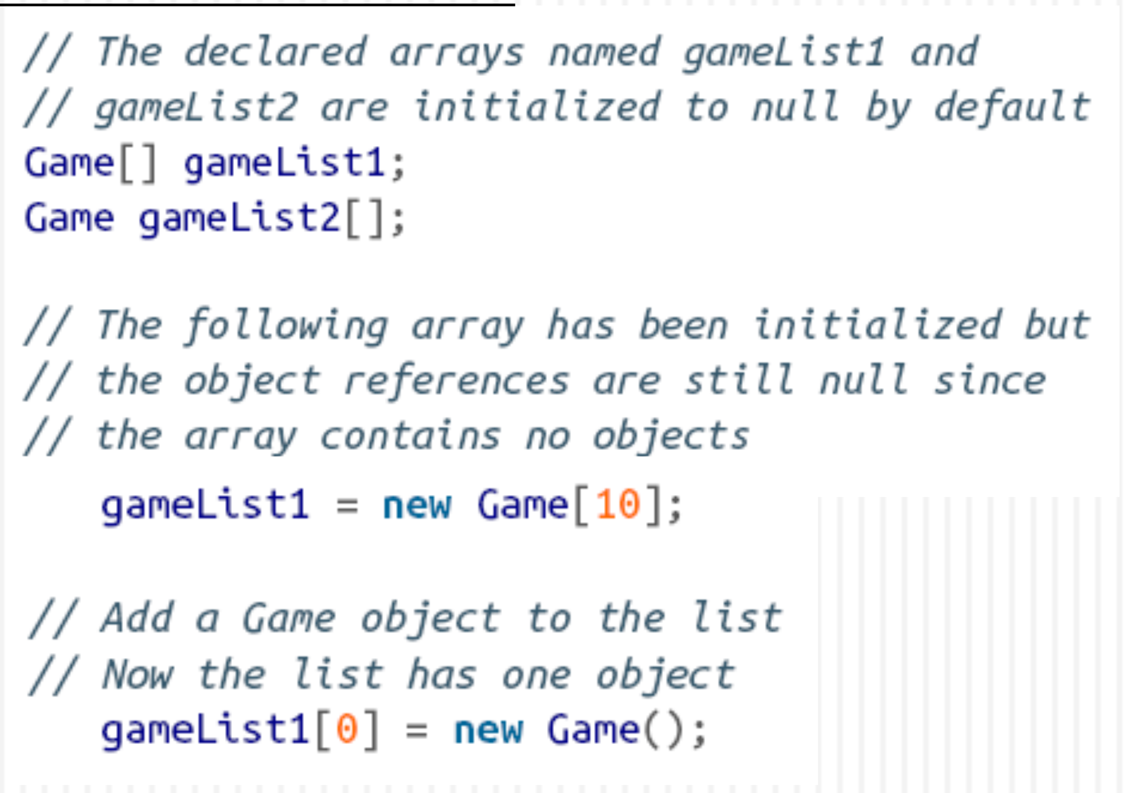
4.3 Array Types



```
public class TestArray {

    public static void main(String[] args) {
        double[] myList = {1.9, 2.9, 3.4, 3.5};

        // Print all the array elements
        for (double element: myList) {
            System.out.println(element);
        }
    }
}
```

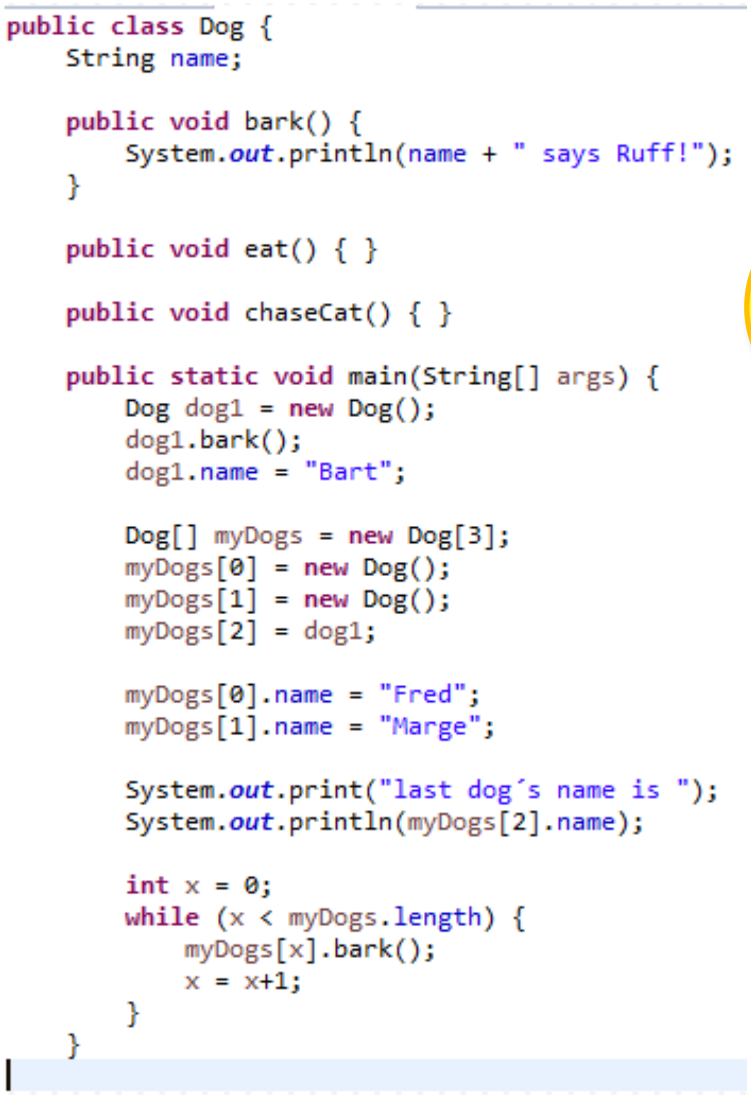
4.3 Array Types

- Arrays are always given a default value whether they are declared as instance variables or local variables. Arrays that are declared but not initialized are given a default value of *null*.

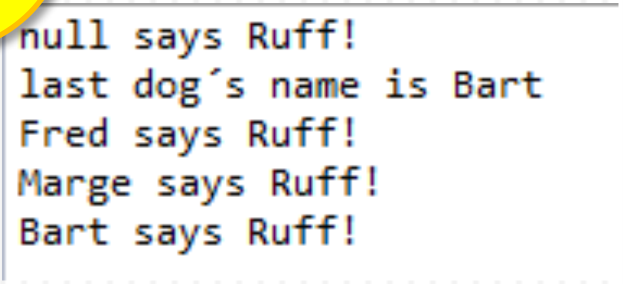


```java
// The declared arrays named gameList1 and
// gameList2 are initialized to null by default
Game[] gameList1;
Game gameList2[];

// The following array has been initialized but
// the object references are still null since
// the array contains no objects
    gameList1 = new Game[10];

// Add a Game object to the list
// Now the list has one object
    gameList1[0] = new Game();
```

4.3 Array Types



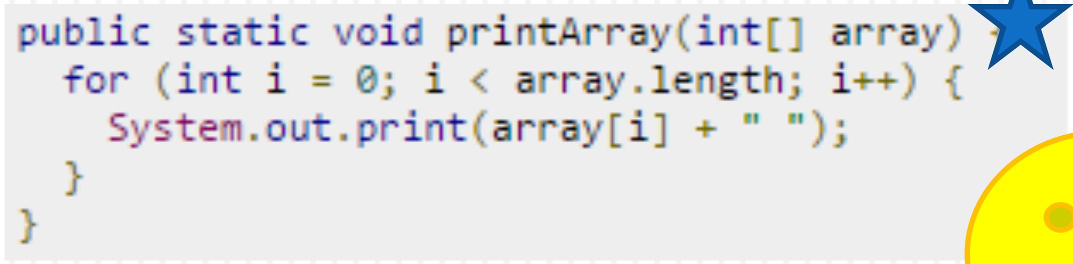
```java
public class Dog {
    String name;

    public void bark() {
        System.out.println(name + " says Ruff!");
    }

    public void eat() { }

    public void chaseCat() { }

    public static void main(String[] args) {
        Dog dog1 = new Dog();
        dog1.bark();
        dog1.name = "Bart";

        Dog[] myDogs = new Dog[3];
        myDogs[0] = new Dog();
        myDogs[1] = new Dog();
        myDogs[2] = dog1;

        myDogs[0].name = "Fred";
        myDogs[1].name = "Marge";

        System.out.print("last dog´s name is ");
        System.out.println(myDogs[2].name);

        int x = 0;
        while (x < myDogs.length) {
            myDogs[x].bark();
            x = x+1;
        }
    }
```



```
null says Ruff!
last dog´s name is Bart
Fred says Ruff!
Marge says Ruff!
Bart says Ruff!
```

4.3 Array Types

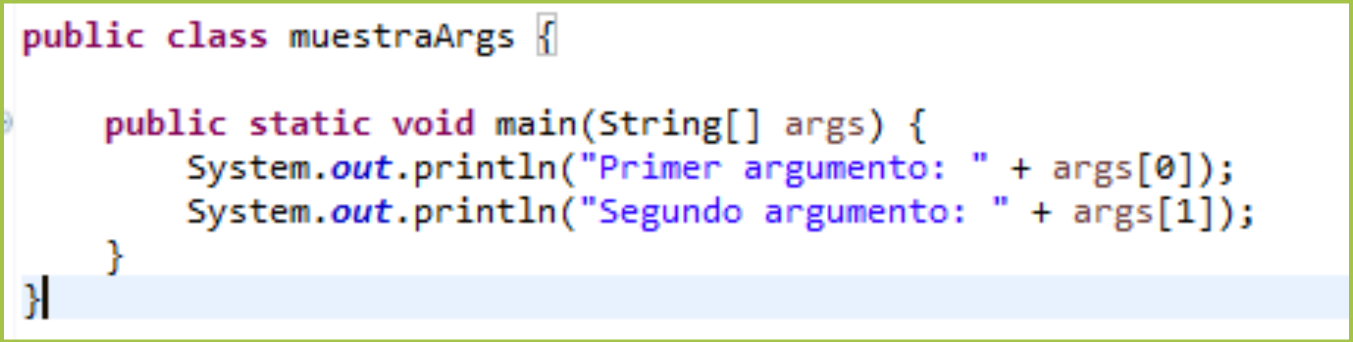
- The length of an array is not part of the array type.
- <u>It is not possible, for example, to declare a method that expects an array of exactly four int values, for example.</u>
- If a method parameter is of type int[], a caller can pass an array with any number of elements.



```
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
    System.out.print(array[i] + " ");
  }
}
```

4.3 Array Types

A.4.1. This program will take from the command line the first two arguments… Make it run!!



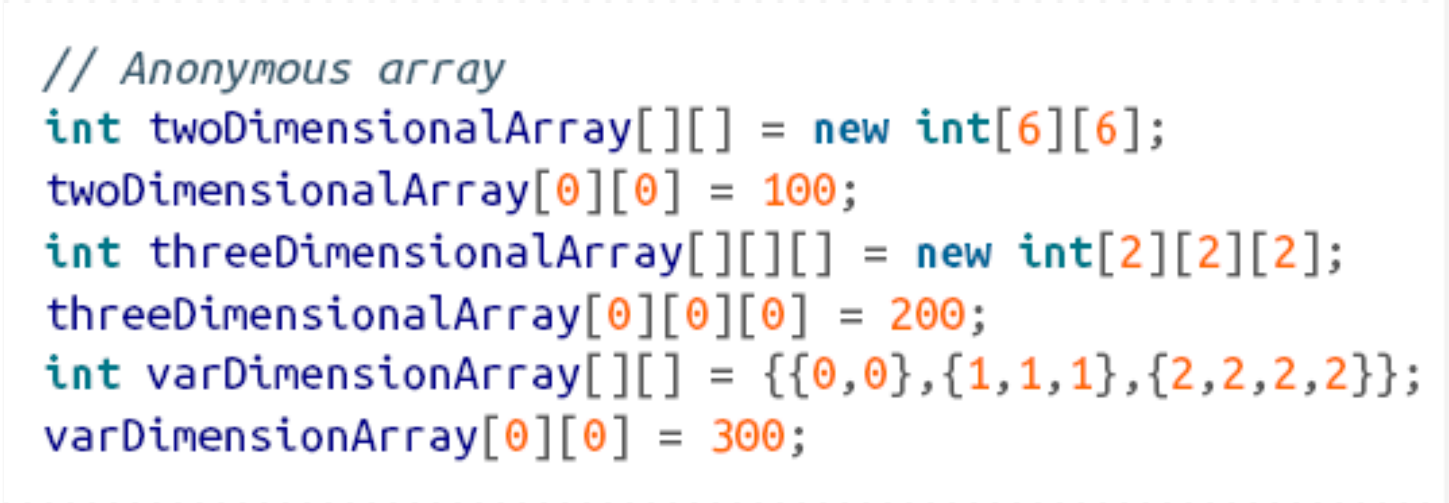
```
public class muestraArgs {

    public static void main(String[] args) {
        System.out.println("Primer argumento: " + args[0]);
        System.out.println("Segundo argumento: " + args[1]);
    }
}
```

4.3 Array Types

A.4.2. Make the same program as the one in slide number 10, with the difference that now you will reverse str using an array. Clue: convert str into an array of Strings.

4.4 Multidimensional arrays

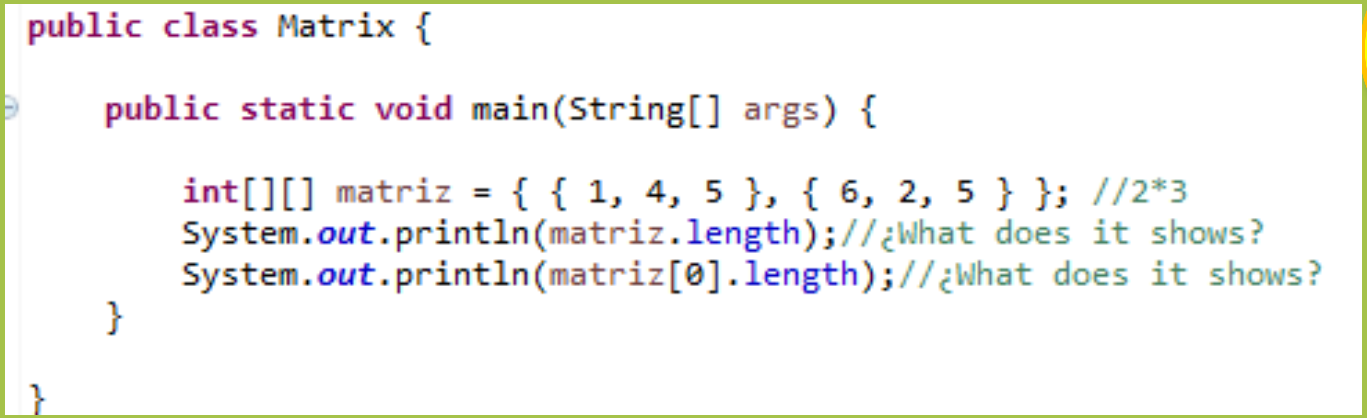
- Multidimensional arrays in Java are actually arrays of arrays.
- They may be initialized with the new operator or by placing their values within braces. Multidimensional arrays may be uniform or non-uniform in shape:



```java
// Anonymous array
int twoDimensionalArray[][] = new int[6][6];
twoDimensionalArray[0][0] = 100;
int threeDimensionalArray[][][] = new int[2][2][2];
threeDimensionalArray[0][0][0] = 200;
int varDimensionArray[][] = {{0,0},{1,1,1},{2,2,2,2}};
varDimensionArray[0][0] = 300;
```

4.4 Multidimensional arrays

A.4.3. Comment what is printed. Afterwards, code in order the program to print the matrix dinamically using loops:



```
public class Matrix {

    public static void main(String[] args) {

        int[][] matriz = { { 1, 4, 5 }, { 6, 2, 5 } }; //2*3
        System.out.println(matriz.length);//¿What does it shows?
        System.out.println(matriz[0].length);//¿What does it shows?
    }

}
```

A.4.4. Write a program where you create a matrix (5x8), and print its elements in a matrix way.

4.4 Multidimensional arrays

A.4.5. Create a new class called Temperatures:

1) Define an attribute called *temp*, which will be an array (5x5) of type int.

2) Create a method called *assignTemperatures*, that will generate random temperatures and fill all the positions from the *temp* array(values between 0 and 50).

3) Crete a method called *calculateAverage*, that will calculate the average value of the values held inside the array. This method has to return the value. Take into account that the result can be a floating number.

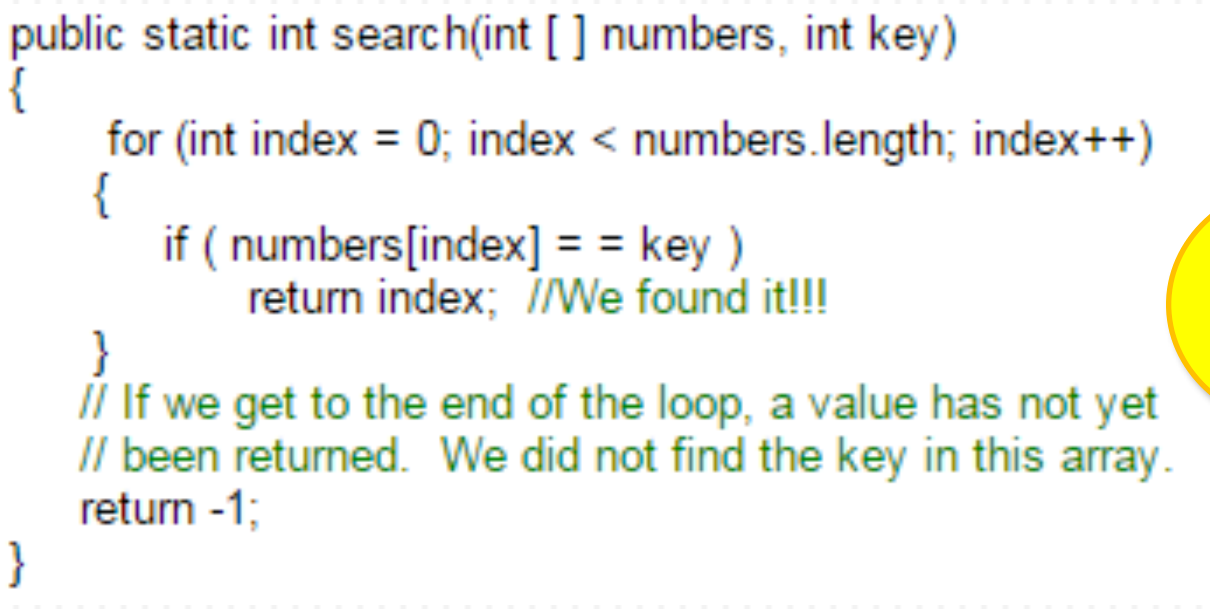
4) Create a method called *printTemperatures* that will print the contenten of *temp*.

4.5 Sequential search

- The simplest type of search is the sequential search.
  - Each element of the array is compared to the key, in the order it appears in the array, until the desired element is found.
  - If you are looking for an element that is near the front of the array, the sequential search will find it quickly. The more data that must be searched, the longer it will take to find the data that matches the key.

4.5 Sequential search

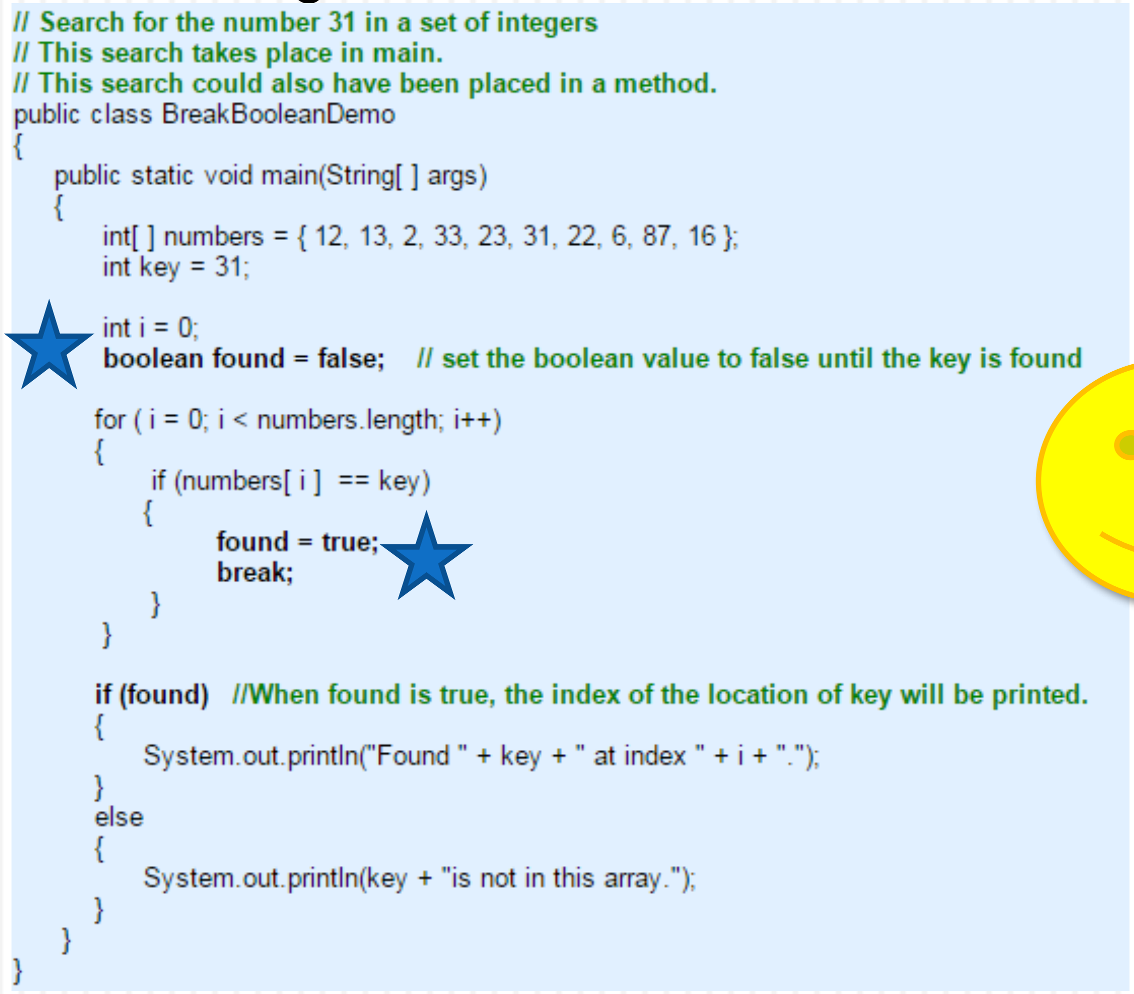
- Consider this method which will search for a key integer value. If found, the index of the first location of the key will be returned. If not found, a value of -1 will be returned.



```
public static int search(int [ ] numbers, int key)
{
        for (int index = 0; index < numbers.length; index++)
       {
            if ( numbers[index] = = key )
                   return index;  //We found it!!!
       }
      // If we get to the end of the loop, a value has not yet
      // been returned.  We did not find the key in this array.
      return -1;
}
```

4.5 Sequential search

- Searching with BREAK and boolean:



```
// Search for the number 31 in a set of integers
// This search takes place in main.
// This search could also have been placed in a method.
public class BreakBooleanDemo
{
    public static void main(String[ ] args)
    {
         int[ ] numbers = { 12, 13, 2, 33, 23, 31, 22, 6, 87, 16 };
         int key = 31;

         int i = 0;
         boolean found = false;    // set the boolean value to false until the key is found

        for ( i = 0; i < numbers.length; i++)
        {
             if (numbers[ i ]  == key)
            {
                   found = true;
                   break;
             }
         }

        if (found)   //When found is true, the index of the location of key will be printed.
        {
            System.out.println("Found " + key + " at index " + i + ".");
        }
        else
        {
            System.out.println(key + "is not in this array.");
        }
    }
}
```

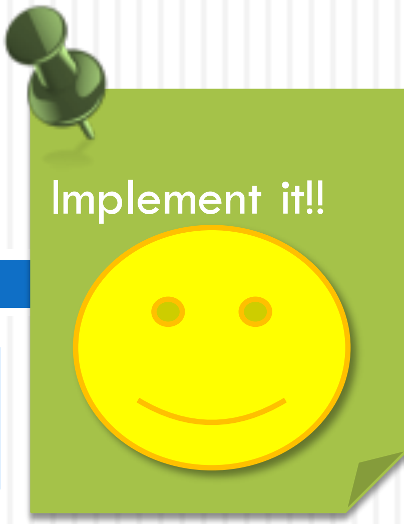
4.6 Binary search

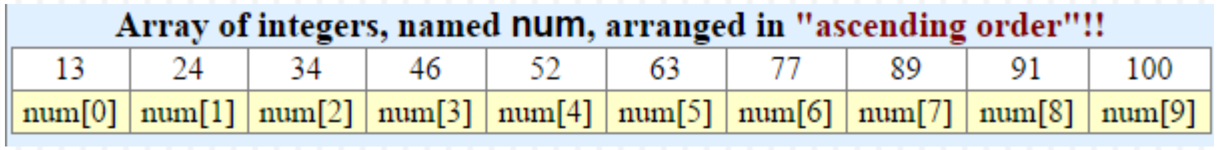
- Do you remember playing the game "Guess a Number", where the responses to the statement "I am thinking of a number between 1 and 100" are "Too High", "Too Low", or "You Got It!"?
- A strategy that is often used when playing this game is to divide the intervals between the guess and the ends of the range in half. This strategy helps you to quickly narrow in on the desired number.
- When searching an array, the binary search process utilizes this same concept of splitting intervals in half as a means of finding the "key" value as quickly as possible.

4.6 Binary search

- If the array that contains <u>your data is in order</u> (ascending or descending), you can search for the key item much more quickly by using a binary search algorithm ("divide and conquer"):
  - In each step, the algorithm compares the search key value with the key value of the middle element of the array.
  - If the keys match, then a matching element has been found and its index, or position, is returned.
  - Otherwise, if the search key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the search key is greater, on the sub-array to the right.
  - If the remaining array to be searched is empty, then the key cannot be found in the array and a special "not found" indication is returned.

4.6 Binary search





Array of integers, named num, arranged in "ascending order"!!

13	24	34	46	52	63	77	89	91	100
num[0]	num[1]	num[2]	num[3]	num[4]	num[5]	num[6]	num[7]	num[8]	num[9]

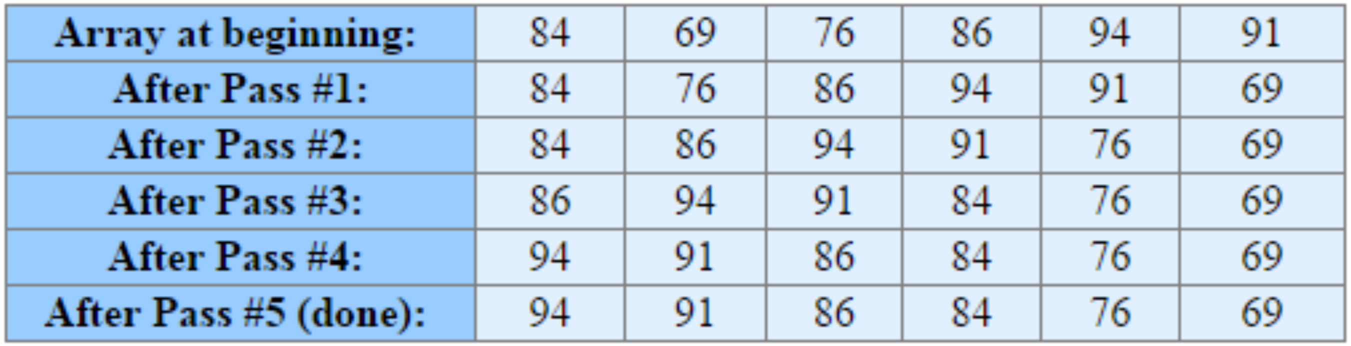
- First, find the middle of the array by adding the array position of the first value to the position of the last value and dividing by two: (0 + 9) / 2 = 4. Integer division is used to arrive at the 4th position as the middle.
- The 4th position holds the integer 52, which comes before 77. We know that 77 will be in that portion of the array to the right of 52. We now find the middle of the right portion of the array by using the same approach. (5 + 9) / 2 = 7
- The 7th subscript holds the integer 89, which comes after 77. Now find the middle of the portion of the array to the right of 52, but to the left of 89. (5 + 6) / 2 = 5
- The 5th subscript holds the integer 63, which comes before 77, so we subdivide again.
- (6 + 6) / 2 = 6 and the 6th subscript holds the integer 77.

4.6 Binary search

A.4.6. Would you be able to implement it?

4.7 Bubble sort

- The bubble sort repeatedly compares adjacent elements of an array.
  - The first and second elements are compared and swapped if out of order.
  - Then the second and third elements are compared and swapped if out of order.
  - This sorting process continues until the last two elements of the array are compared and swapped if out of order.
  - Example descending order:



Array at beginning:	84	69	76	86	94	91
After Pass #1:	84	76	86	94	91	69
After Pass #2:	84	86	94	91	76	69
After Pass #3:	86	94	91	84	76	69
After Pass #4:	94	91	86	84	76	69
After Pass #5 (done):	94	91	86	84	76	69

4.7 Bubble sort

A.4.7. Write a program where you have to create an array of int type and length 50. Its values will be integers between 1 and 100. Afterwards, you have to arrange them in **descending** order using the Bubble algorithm and show them in order to through the console.

4.8 Insertion sort

- The insertion sort splits an array into two sub-arrays.
  - The first sub-array is always sorted and increases in size as the sort continues.
  - The second sub-array is unsorted, contains all the elements yet to be inserted into the first sub-array, and decreases in size as the sort continues.
- Let's look at an same example using the insertion sort for descending order:

Array at beginning:	84	69	76	86	94	91
🟩 = 1st sub-array	84	69	76	86	94	91
🟪 = 2nd sub-array	84	69	76	86	94	91
	84	76	69	86	94	91
	86	84	76	69	94	91
	94	86	84	76	69	91
2nd sub-array empty	94	91	86	84	76	69

4.8 Insertion sort

A.4.8. Create a program where you will have two vectors of length 100. The first one will hold numbers generated randomly. In the second array you will have to get the same data but order them using the *insertion sort* method.