Advanced Programming Crash Course

By Shahrukh Qureshi

What will we learn?

- Arrays
- 2D Arrays (Matrices)
- Array Sorting
- Collection Classes
- Recursion

Arrays

- Location in memory that stores multiple elements under the same TYPE and NAME

```
// Ways to declare an array
String stringArray[] = new String[2];
String stringArray2[] = new String[] { "Element 1", "Element 2" };
String stringArray3[];
stringArray3 = new String[2];

System.out.println("Method 1 of outputting using a loop");
for (int i = 0; i < stringArray2.length; i++) {
    System.out.println(stringArray2[i]);
}

System.out.println("Method 2 of outputting using a loop");
for (String string : stringArray2) {
    System.out.println(string);
}
```

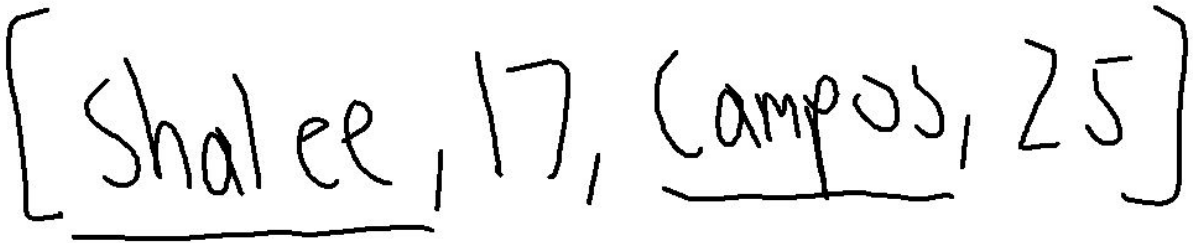
2D Arrays (Matrices)

- Some data is stored in multiple dimensions
  - Example: Tables, Maps, Images

Name	Age
Shalee	17
Campos	25

2D Arrays (Matrices) – Continued

- A lot harder to manipulate the data if its in a 1D array
- We need to call each row by 2 indices

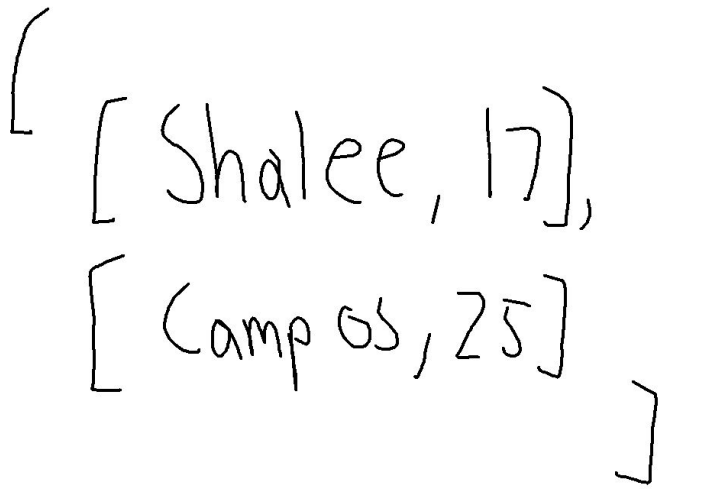


In this example:

Row 1 is index 0 and index 1

2D Arrays (Matrices) – Continued

- Easier if its 2D since we can call each Row by 1 index



Shalee	17
Campos	25

In this example:

Row 1 is index 0

2D Arrays (Matrices) – Continued

- Index order for an individual element is [Row][Col]

Name	Age
Shalee	17
Campos	25

Name	Age
Row 1, Col 1 (0, 0)	Row 1, Col 2 (0, 1)
Row 2, Col 1 (1, 0)	Row 2, Col 2 (1, 1)

2D Arrays (Matrices) – Continued

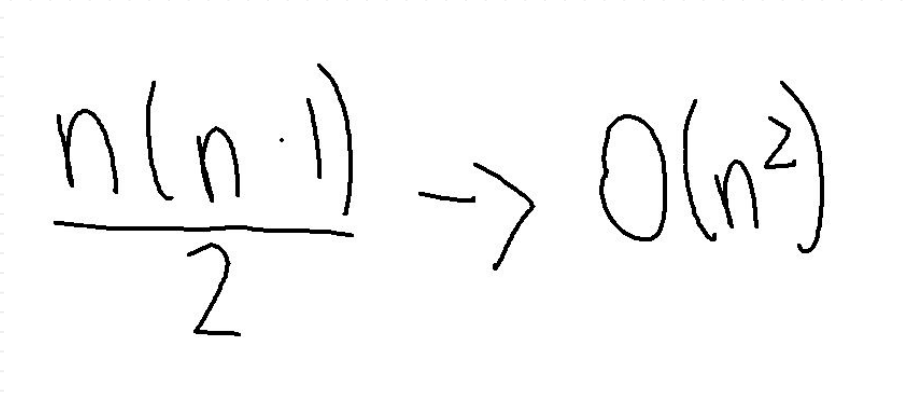
```java
// Ways to declare a 2D array
int intArray[][] = new int[2][2];
int intArray2[][] = new int[][] { { 1, 2, 3 }, { 4, 5, 6 } };
int intArray3[][];
intArray3 = new int[2][2];

System.out.println("Method 1 of outputting using a loop");
for (int i = 0; i < intArray2.length; i++) {
    for (int j = 0; j < intArray2[i].length; j++) {
        System.out.print(intArray2[i][j] + " ");
    }
    System.out.println();
}

System.out.println("Method 2 of outputting using a loop");
for (int[] is : intArray2) {
    for (int is2 : is) {
        System.out.print(is2 + " ");
    }
    System.out.println();
}
```

Array Sorting

- Array Sorting is crucial to organizing data
  - Ascending order, Descending order, Alphabetical, etc
- Algorithms we will discuss
  - Bubble Sort
  - Selection Sort
  - Insertion Sort

Array Sorting – Continued

- Bubble Sort
  - Compare the first two elements
    - Swap if out of order
    - Continue this until you reach the end of the array

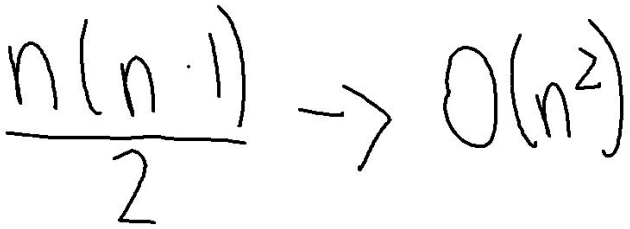


Array Sorting – Continued

- Example: Sorting in Ascending order
  - Array: [5, 3, 8, 4, 6]
    - [5, 3, 8, 4, 6] -> Compare first and second (Swap)
    - [3, 5, 8, 4, 6] -> Compare second and third (Don't swap)
    - [3, 5, 8, 4, 6] -> Compare third and fourth (Swap)
    - [3, 5, 4, 8, 6] -> Compare fourth and fifth (Swap)
    - [3, 5, 4, 6, 8] -> Repeat this process until no more swaps are required

Array Sorting – Continued

- Selection Sort
  - Search for the smallest element and swap it into the first index
  - Search for the second smallest element and swap it into the second index
  - Continue this process until the array is fully sorted



Array Sorting – Continued

- Insertion Sort
  - Takes advantage of pre-sorting (only 1 pass)
  - Requires fewer comparisons than the other 2 Algorithms
  - Merge based (merging unsorted data to the sorted side)
- View pseudocode here: Click me

Array Sorting – Continued

		Time Complexity					
		Best	Worst	Avg.	Space	Stable	Comments
Comparison Sort	Bubble Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Yes	*For each pair of indices, swap the elements if they are out of order*
	Modified Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	*At each Pass check if the Array is already sorted. Best Case-Array Already sorted*
	Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Yes	*Swap happens only when once in a Single pass*
	Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Yes	*Very small constant factor even if the complexity is O(n^2).* ***Best Case:*** *Array already sorted* ***Worst Case:*** *sorted in reverse order*
	Quick Sort	O(n.lg(n))	O(n^2)	O(n.lg(n))	O(1)	Yes	*Best Case: when pivot divide in 2 equal halves Worst Case: Array already sorted - 1/n-1 partition*
	Randomized Quick Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(1)	Yes	*Pivot chosen randomly*
	Merge Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(n)	Yes	*Best to sort linked-list (constant extra space). Best for very large number of elements which cannot fit in memory (External sorting)*
	Heap Sort	O(n.lg(n))	O(n.lg(n))	O(n.lg(n))	O(1)	No	
Non-Comparison Sort	Counting Sort	O(n+k)	O(n+k)	O(n+k)	O(n+2^k)	Yes	*k = Range of Numbers in the list*
	Radix Sort	O(n.k/s)	O(2^s.n.k/s)	O(n.k/s)	O(n)	No	
	Bucket Sort	O(n.k)	O(n^2.k)	O(n.k)	O(n.k)	Yes	

Collection Classes

- Lists
  - An ordered collection of elements
    - Elements can be accessed through indices
    - May Contain duplicate values
- Sets
  - An unordered collection of elements
    - No indices
    - All values must be unique

View code

Recursion

- An Algorithm where a method calls itself
  - Form of looping where a method calls itself repeatedly to solve simpler versions of a problem
- Properties
  - Must contain a base case
    - Has to be reached eventually to prevent a StackOverFlow error
  - Contains decision structures rather than conventional loops

Recursion – Continued

- Process
  - Involves the usage of a Stack
    - Stack = Data Abstraction where:
      - New data is “pushed” to the top of the stack
      - Old data is “popped” or removed from the top of the stack
  - The current computation is suspended and placed onto a Stack
  - Once the base case is reached the method unwinds and the Stack is popped

Recursion – Continued Part 2

```java
private static int power(int base, int n) {

    // If the exponent (n) is less than or equal to 1 we return the base
    if (n <= 1) {
        return base;
    }
    // If the exponent (n) is greater than 1 we recursively call this method
    return base * power(base, n - 1);
}

Run | Debug
public static void main(String[] args) {
    int value = power(2, 3); // Calling the power method
    System.out.println(value); // Outputting the answer
}
```

View PPT Code