Arrays

❖ Definition: a collection of data values of the same "type"

"type": any primitive data type (i.e., int, float, boolean, char, etc.)
e.g., a collection of 10 gpa's (doubles)

OR

any reference data type (i.e., objects)
e.g., a collection of 20 Student objects

NOTE: An array is an object. (can have an array of arrays!)

❖ Declaration: **<type> [] <name>;**

<type>: the data type of each element of the array
[]: the index operator (has highest precedence of all operators)
<name>: any valid Java identifier

Examples:

```
String[] args;     // an array of Strings
int[] intArray;    // an array of integers
double[] gpaList;  // an array of doubles
```

❖ Alternate Declaration: **<type> <name> [];**

Not as commonly used.
Not consistent with other object declarations.

Arrays…continued

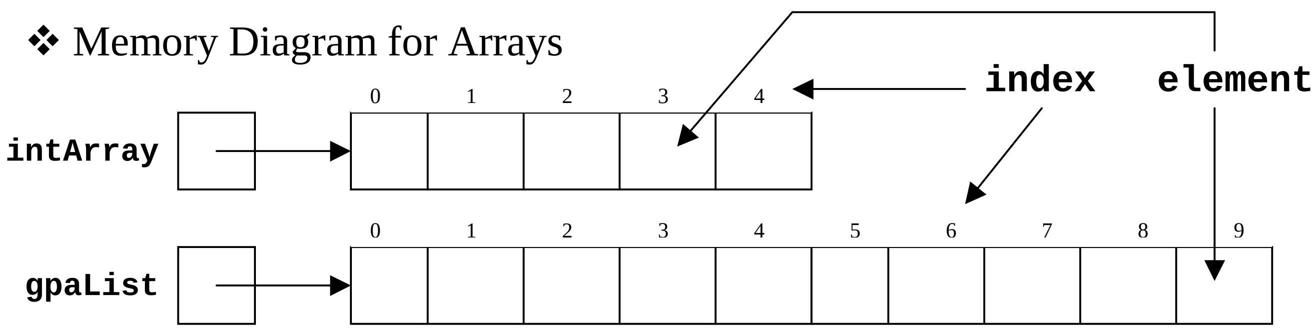
❖ Initialization: **<name> = new <type> [size];**

size: the number of elements in the array (aka "**length**")

Examples:

```
intArray = new int[5];
gpaList = new double[10];
```

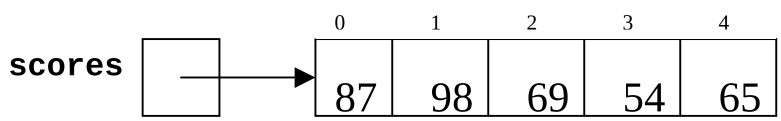
❖ Memory Diagram for Arrays



NOTE: After initializing the array, the elements remain uninitialized

❖ Initializer List ("Shortcut" Declaration and Initialization):

e.g., `int[] scores = {87, 98, 69, 54, 65};`



NOTE:

- No '**new**' operator
- No size indicated (the # of elements within the braces {}, separated by commas, determines the length of the array)
- Initializer lists can ONLY be used when array is FIRST declared. i.e., the following is INVALID:

```
int[] scores;
scores = {87, 98, 69, 54, 65};
```

Arrays…continued

❖ Accessing the elements of an array

- use the index [] operator
- elements are specified using a numeric index starting at zero
- the index must be between 0 and the length of the array-1 ("off-by-one" errors are common—watch out!)
- attempting to access an element of an array that does not exist will cause an error (the compiler does "Bounds Checking" and may throw an ArrayIndexOutOfBoundsException)

- Syntax: **`<name> [<index>]`**

e.g.,
```
intArray[3] = 5;
int i = intArray [3];
```

❖ Array length

- Arrays have a constant public data member called **`length`**
- An array's length cannot be changed once the array has been instantiated
- The **`length`** data member in arrays is DIFFERENT than the method **`length()`** in Strings
- Code to initialize all the elements of an array:

```
for (int i = 0; i < array.length; i++) {
    array[i] = <some value>;
}
```

`array[i]` is used as the "identifier" of the elements within the array

- The length of the array need not be "hard-coded", but rather could be determined at run-time:

```
//size determined at run-time (e.g. user input)
int[] intArray = new int[size];
```

Arrays…continued

Write a method to print to standard output ALL the values within an array of integers that is taken as a parameter.

```
public static void printArray (int[] array) {
```

Arrays…continued

Write a method to search for a particular value within an array of integers that is taken as a parameter. The method should return the index of the array containing the value (or –1 if the value is not found). The value to look for should also be taken as a parameter.

```
public static int searchArray (int[] array, int val) {
```

Arrays…continued

Write a method that swaps the element at index **`a`** with the element at index **`b`** of an array of integers that is taken as a parameter. Assume that **`a`** & **`b`** are valid indices of the array, and they are also taken as parameters.

```
public static void swap (int[] array, int a, int b) {
```

Arrays…continued

Write a method that returns the index of the smallest element of an array of integers taken as a parameter. The search for the smallest element should start at the index **`start`** and continue to the end of the array.

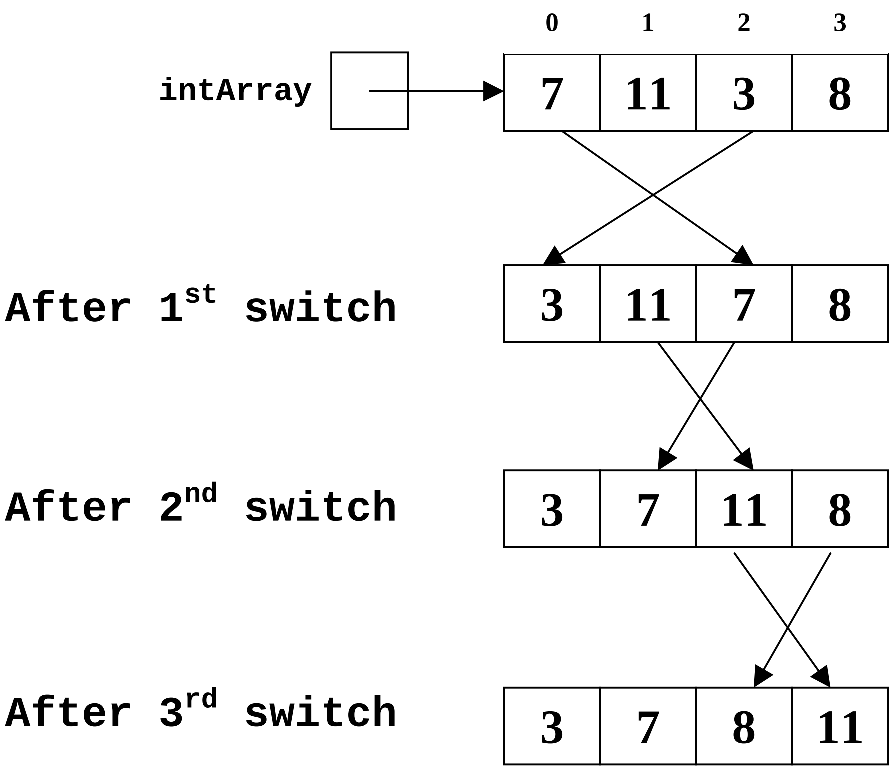
Do a trace of the following array passed to the min method:
{8,4,7,2,7,5}; start the search at element 0.

```
public static int min (int[] array, int start) {
```

Write a method that will sort an array using the swap and min methods (i.e., a “Selection Sort”)

❖ <u>Selection Sort Algorithm</u>

1. Divide the array into two sections:
   - an unsorted section (initially the whole array)
   - sorted section (initially “empty”)
2. With each step, move the smallest value from the unsorted section to the end of the second section



❖ CODE: (Uses the **`swap()`** and **`min()`** methods defined above)

```
public static void sort (int[] array) {
```

Arrays…continued

Working with arrays of objects:

❖ Declaration:
```
<class name> [] <name>;
```

Example:
```
Fish[] fishArray;  // an array of Fish
```

❖ Initialization:
```
<name> = new <type> [size];
```

Example:
```
fishArray = new Fish[3];
```

❖ Initialization of the elements of the array:

```
<name>[<index>] = new <class name> (<args>);
```

Example:
```
fishArray[0] = new Fish (<args>);
```

❖ Initializer List (Shortcut Declaration and Initialization):

e.g.,
```
Fish[] fishArray = {new Fish(<args>),
    new Fish(<args>), new Fish(<args>)};
```

❖ Accessing the elements' methods:

```
<name>[<index>].<method name> (<args>);
```

Example:
```
fishArray[0].setPosition (newPosition);
```

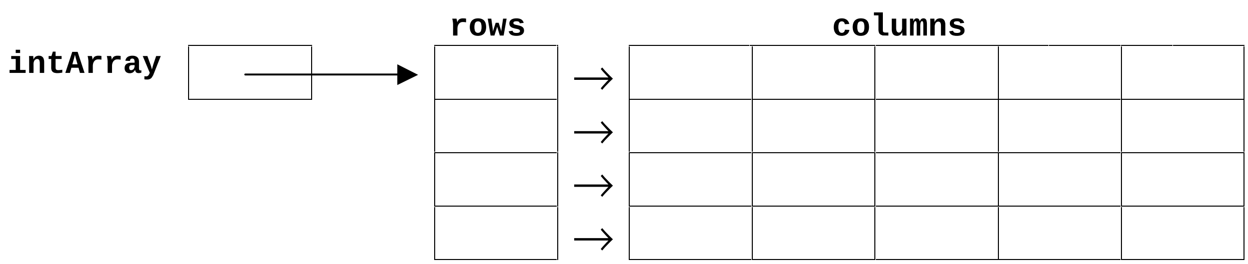
Multi-Dimensional Arrays

An array can contain elements of ANY type, including arrays. Therefore, a "two-dimensional" array can be created. (In fact, multi-dimensional arrays can be developed!)

Declaration: **<type>[][] <name>; // 2D array**

Initialization: **<name> = new <type>[<#rows>][<#cols>];**

Example: **int[][] intArray = new int [4][5];**



OR: A 4 X 5 Table

	0	1	2	3	4
0					
1					
2					
3					

For array lengths:

intArray.length == 4
intArray[0].length == 5

Accessing an element of a two-dimensional array (see Fig 9.18)

<u>Multi-Dimensional Arrays...continued</u>

Arrays with greater than 2 dimensions can be developed, but in object-oriented programming, it is usually better to create an object with multiple one-dimensional arrays as data members.

Example: **`int[][][][] intArray = new int[5][5][5][5];`**

<u>Jagged Arrays</u>: The sub-arrays do NOT necessarily have to be the same size.

Code to initialize a Jagged array:

Algorithm:

- Pass an array of sizes of each sub-array to a method.
- The value of each element is the length of each sub-array (in the 2nd dimension)
- The length of the 1st dimension is the array.length of the array that is passed.

Code:

```
public static int[][] jaggedArray (int[] sizes)
{
    int[][] array = new int[sizes.length][];
    for (int i = 0; i < array.length; i++) {
        array[i] = new int[sizes[i]];
    }
    return array;
}
```

Multi-Dimensional Arrays...continued

Write a method that prints (to standard output) each element of a jagged two-dimensional array. Each sub-array should be separated by a newline and each element of each sub-array should be separated by a space.

e.g.:

```
5 9 4
6 8 2 5 4
1 2 2 3
```

```
public static void print (int[][] array) {
```

Creation of MyArray

Create a class called MyArray that will basically allow an array to change size (i.e., grow and shrink). The array should ONLY hold objects as its elements (i.e., no primitive types). The array will have a capacity (# of elements it can hold) and a size (# of elements is currently is holding).

The following methods should be supported (use the exact method header for each method):

- **`void addElement (Object element)`** – adds an element to the end of the array and adjusts the capacity if necessary

- **`boolean contains (Object element)`** – returns true if the Object **`element`** is contained within the array; otherwise returns false

- **`Object elementAt (int index)`** – returns the element at location **`index`**

- **`void setElementAt (Object element, int index)`** – sets the element at the specified **`index`** equal to Object **`element`**

- **`void removeElementAt (int index)`** – removes element at the specified **`index`**

- **`int size ()`** – returns the number of elements currently in the array

- **`int capacity ()`** – returns the current capacity of the array

What other methods might be helpful?

```
class MyArray {

    // Data Members

    //Constructors

    public MyArray ()

    public MyArray (int initCapacity)

    public MyArray (Object[] a)

    //Methods

    public int size ()

    public int capacity ()

    public Object elementAt (int index)

    public void setElementAt (Object element, int index)

    public void removeElementAt (int index)

    public void addElement (Object element)

    public void addElement (Object element, int index)

    private void increaseCapacity ()

    public boolean contains (Object element)

    public int indexOf (Object element)

    public void clear ()

    public void trimToSize ()

    public boolean isEmpty ()

    public String toString ()

}
```

Conditional Operator

❖ A ternary operator that takes 3 operands

❖ Syntax:

```
<variable> = <boolean expression> ?
             <expression1> :
             <expression2>;
```

equivalent to:

```
if (<boolean expression>) {
    <variable> = <expression1>;
}
else {
    <variable> = <expression2>;
}
```

`<variable>`: any primitive type or reference type variable (the variable can be initialized in this statement)

`<boolean expression>`: any expression that evaluates to **`true`** or **`false`**;

`<expression1>`: any expression that evaluates to the same data type as **`<variable>`**

`<expression2>`: any expression that evaluates to the same data type as **`<variable>`**

❖ Example: Obtaining the minimum between two values:

```
min = x < y ? x : y;
```

equivalent to:

```
if (x < y)
    min = x;
else
    min = y;
```