Revised: 5/7/2025

Arrays I

Michael C. Hackett

Assistant Professor, Computer Science

COMMUNITY COLLEGE OF PHILADELPHIA

Lecture Topics

- Array Basics
  - Declaration/Initialization
  - Retrieving/Changing values of elements
  - Array Length
- Iterating Through an Array
  - Traditional For Loop
  - Enhanced For Loop
- Copying an Array
  - Shallow Copies
  - Deep Copies

Colors/Fonts

- Local Variable Names – Brown
- Primitive data types – Fuchsia
- Literals – Blue
- Keywords – Orange
- Object names – Green
- Operators/Punctuation – Black
- Field Names – Lt Blue
- Method Names – Purple
- Parameter Names – Gold
- Comments – Gray
- Package Names – Pink

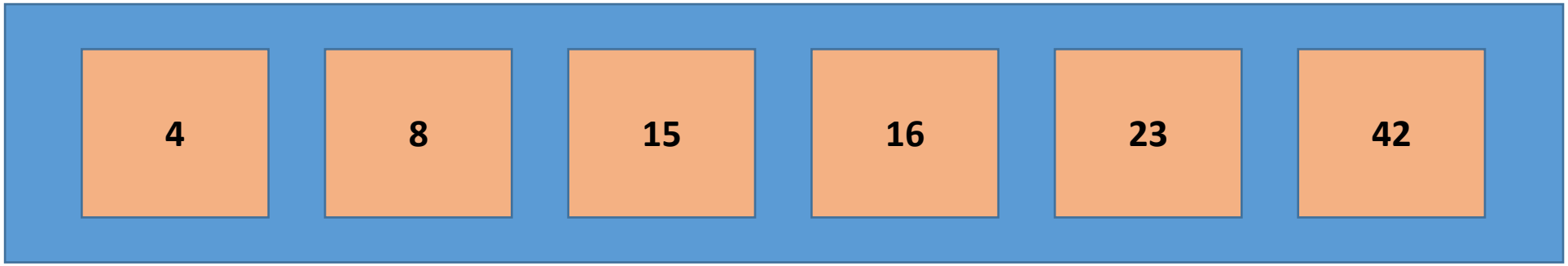
Source Code – **Consolas**
Output – Courier New

Boolean expression is false

Boolean expression is true

Arrays

- An ***array*** is a container object that holds a fixed number of values.
- All values must be of the same data type or object (all ints, Strings, etc.)



An array of ints

Array Basics

- Declared just like any other variable with one difference:
  - An open and close bracket is included after the data type (shown below)

```
int[] numbers;

double[] values;

String[] names;
```

Array Basics

- There are two ways to initialize an array:
  - Without data:

```
numbers = new int[6];
```

  - Or with data:

```
numbers = {4, 8, 15, 16, 23, 42};
```

Values are comma separated in open/close braces

- Has space for 6 ints.
- Default values are
  - 0 (for numeric arrays)
  - ' ' (for char)
  - false (for boolean)
  - null (for objects)

Array Basics

- Without data:

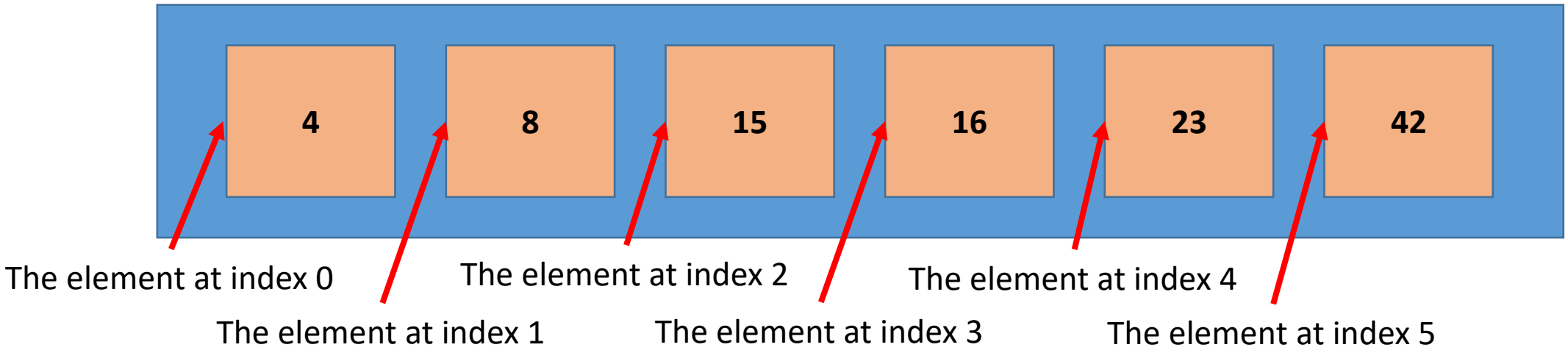
```
int[] numbers = new int[6];
```

- Or with data:

```
int[] numbers = {4, 8, 15, 16, 23, 42};
```

Array Basics

- An ***index*** (or ***subscript***) is the number representing the position of an array element.
  - First index is always zero.
  - The index is always of type int.
- An ***element*** is the data or object referenced by an index.



Array Basics

- Internally, arrays do not technically have a size limit, however…
  - Array indexes are represented by an int.
  - The maximum value of an int is 2,147,483,647.
  - Therefore, that is the maximum length of an array.

Array Basics

- Variables map to locations in memory using a ***symbol table***.
  - A symbol could be a variable or class or method, for example.
  - Managed by the operating system.

- A memory map is a diagram of memory addresses and the data associated with an address or addresses.
  - An address typically corresponds to 8 bits/1 byte of space.

Array Basics

Variables

```
int number = 58;
double value = 16.5;
char letter = 'L';
```

ints are 32 bits/4 bytes
doubles are 64 bits/8 bytes
chars are 16 bits/2 bytes

Symbol Table

Symbol	Address
number	1000
value	1004
letter	100C

Memory Map

Address	Data
1000	58
1001	
1002	
1003	
1004	16.5
1005	
1006	
1007	
1008	
1009	
100A	
100B	
100C	L
100D	

Array Basics

Variables

```
char letter = 'L';
int[] numbers = {5, 3, 1}
```

Symbol Table

Symbol	Address
letter	1000
numbers	1002

Memory Map

Address	Data
1000	L
1001	
1002	numbers[0]
1003	5
1004	
1005	
1006	numbers[1]
1007	3
1008	
1009	
100A	numbers[2]
100B	1
100C	
100D	

Arrays

- The array's ***base address*** is the address where the array begins.

- The following formula calculates the address of other indexes in the array:

base address + index * byte size

Array Basics

Symbol Table

Symbol	Address
letter	1000
numbers	1002

Memory Map

Address	Data
1000	**L**
1001	
1002	numbers[0]
1003	**5**
1004	
1005	
1006	numbers[1]
1007	**3**
1008	
1009	
100A	numbers[2]
100B	**1**
100C	
100D	

numbers[0] = 1002_{16} + 0 * 4 = 1002_{16}

numbers[1] = 1002_{16} + 1 * 4 = 1006_{16}

numbers[2] = 1002_{16} + 2 * 4 = $100A_{16}$

- Memory addresses are represented using the hexadecimal system (Base-16).

Array Basics

- After initializing an empty array…

```
int[] numbers = new int[6];
```

- You can initialize the elements by referencing the index using ***subscript notation***:

```
numbers[0] = 4;
numbers[1] = 8;
numbers[2] = 15;
numbers[3] = 16;
numbers[4] = 23;
numbers[5] = 42;
```

Array Basics

- To retrieve an array's element, simply reference the index:

```
int[] multiplesOfTen = {10, 20, 30, 40, 50};

System.out.println(multiplesOfTen[0]);
System.out.println(multiplesOfTen[1]);
System.out.println(multiplesOfTen[2] + multiplesOfTen [3]);
```

```
10
20
70
```

Array Basics

- Assign a new value/object to the array at the desired index.

```
int[] myTwoNumbers = new int[2];
myTwoNumbers[0] = 10;
myTwoNumbers[1] = 20;
System.out.println(myTwoNumbers[0]);
System.out.println(myTwoNumbers[1]);

myTwoNumbers[0] = 30;
myTwoNumbers[1] = 40;
System.out.println(myTwoNumbers[0]);
System.out.println(myTwoNumbers[1]);
```

```
10
20
30
40
```

Array Basics

- To retrieve an array's length, call on it's length field.
  - The length field is an int.

```
int[] numbers = new int[10];
int numbersLength = numbers.length;
System.out.println(numbersLength);
```

```
10
```

ArrayIndexOutOfBoundsException

- This exception is caused by:
  - Trying to retrieve a value at an a non-existent index.
  - Trying to store a value to a non-existent index.
  - Using a negative as an index (`someArray[-1]`)

```
char[] letters = {'a', 'b', 'c'};
System.out.println(letters[3]);
```

```
run:
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
        at javaapplication3.JavaApplication3.main(JavaApplication3.java:23)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Iterating Through an Array

- For loops are ideal for iterating through the elements of an array.
  - The loop's counter can be used to represent each index.

```
String[] names = {"John", "Jane", "Jack"};

for(int i = 0; i < names.length; i++) {
   System.out.println(names[i]);
}
```

```
John
Jane
Jack
```

This loop will iterate from 0 through 2.

Will be 0, then 1, then 2

Iterating Through an Array

- This for loop demonstrates the ability to initialize the elements of an array.

```
String[] names = new String[3];
Scanner keyboard = new Scanner(System.in);
for(int i = 0; i < names.length; i++) {
    System.out.print("Enter name #" + i + ": ");
    names[i] = keyboard.nextLine();
}

//Prints the values of the names array
for(int i = 0; i < names.length; i++) {
    System.out.println(names[i]);
}
```

```
Enter name #0: John
Enter name #1: Jane
Enter name #2: Jack
John
Jane
Jack
```

Iterating Through an Array

- You may have noticed the output started by asking for name #0.
  - It would look better if it started by asking for name #1
  - We would still want to assign that name to index 0, though.

```
Enter name #0: John
Enter name #1: Jane
Enter name #2: Jack
John
Jane
Jack
```

Iterating Through an Array

- This change will add one to i when printed.
  - But it won't actually replace the current value of i.

```
String[] names = new String[3];
Scanner keyboard = new Scanner(System.in);
for(int i = 0; i < names.length; i++) {
   System.out.print("Enter name #" + (i + 1) + ": ");
   names[i] = keyboard.nextLine();
}
```

```
Enter name #1: John
Enter name #2: Jane
Enter name #3: Jack
```

Iterating Through an Array

- This for loop demonstrates the ability to change or alter the values of an array.

```
String[] names = {"John", "Jane", "Jack"};

for(int i = 0; i < names.length; i++) {
   names[i] = names[i].toUpperCase();
}
//Prints the values of the names array
for(int i = 0; i < names.length; i++) {
   System.out.println(names[i]);
}
```

Replaces the original value with an uppercase version of itself

```
JOHN
JANE
JACK
```

Iterating Through an Array

- This for loop iterates through the elements backwards.

```
String[] names = {"John", "Jane", "Jack"};

for(int i = names.length - 1; i >= 0; i--) {
   System.out.println(names[i]);
}
```

Starts at the last index

Stops when -1 is reached

```
Jack
Jane
John
```

Iterating Through an Array

- This for loop iterates through a portion of the array.

```
String[] names = {"John", "Jane", "Joe", "Jack"};

for(int i = 0; i < names.length/2; i++) {
   System.out.println(names[i]);
}
```

```
John
Jane
```

Iterating Through an Array

- The ***for-each loop*** (also known as ***enhanced for loop*** or ***for-in loop***) is special type of for loop that iterates over the contents of an array or list.
  - This is the type of for loop Python uses.

```
for(dataType variableName : arrayName) {
   ...
}
```

- For each element in the array or list, *variableName* will represent that element for each iteration.
  - The data type of variableName must match the data type of the array.

Iterating Through an Array

```
String[] names = {"John", "Jane", "Joe", "Jack"};

for(String name : names) {
   System.out.println(name);
}
```

```
John
Jane
Joe
Jack
```

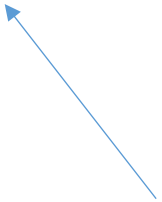
Iterating Through an Array

- For-each loops will iterate over the entire length of the array.
  - Even if there is no element present.

```
String[] names = new String[3];
names[0] = "John";
names[1] = "Jane";

for(String name : names) {
   System.out.println(name);
}
```

```
John
Jane
null
```



No String stored at index 2

Iterating Through an Array

- For-each loops are the preferred way to iterate over every element, from start to finish.

- Benefits of the for-each loop :
  - No need to worry about array size/length.
  - No need to worry about any ArrayIndexOutOfBoundsExceptions.

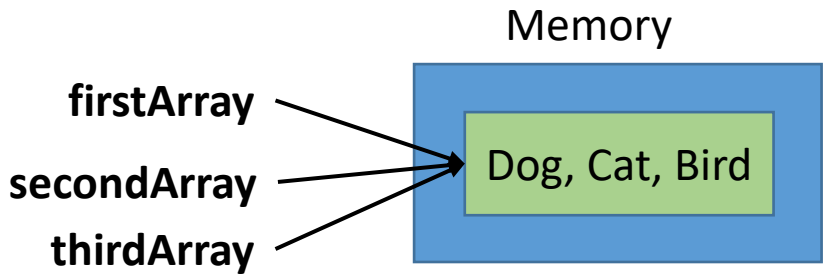
- Drawbacks of the for-each loop:
  - Can’t change the elements in the array.
  - Can’t go in reverse.
  - Can’t iterate over a portion of the array.
  - Can’t work with additional arrays in the loop.
    - For example, copying elements from one array to another.
  - Doesn’t keep track of subscripts/index numbers.
    - There’s no counter variable like a traditional for loop.

Copying an Array

- Copying an array like the example below creates a ***shallow copy.***
  - Shallow copies are multiple variables referencing the same data.

```
String[] firstArray = {"Dog", "Cat", "Bird"};
String[] secondArray = new String[5];

secondArray = firstArray;
String[] thirdArray = firstArray;
```



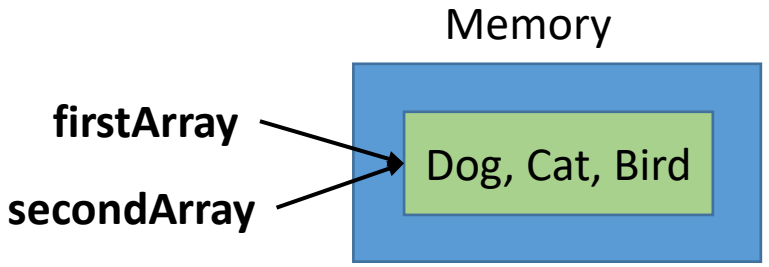
Copying an Array

- Since the variables reference the same array, changing one appears to change any others.

```
String[] firstArray = {"Dog", "Cat", "Bird"};
String[] secondArray = new String[5];
System.out.println(firstArray[0]);

secondArray[0]= "Fish";
System.out.println(firstArray[0]);
```

```
Dog
Fish
```



Copying an Array

- To create a second, separate array with the same contents you need to perform a ***deep copy***.
  - A deep copy copies the contents of one array into a second array of the same length.

```
int[] original = {3, 5, 7, 9};
int[] copy = new int[original.length];

for(int i = 0; i < original.length; i++) {
   copy[i] = original[i];
}
```

Same size

Copies the value to the corresponding index in the other array.

Copying an Array

- Since the variables reference different arrays, changing one does not alter the original.

```
int[] original = {3, 5, 7, 9};
int[] copy = new int[original.length];

for(int i = 0; i < original.length; i++) {
    copy[i] = original[i];
}

System.out.println(original[0]);
copy[0]= 99;
System.out.println(original[0]);
```

```
3
3
```