Arrays

Arrays are not part of the Collection classes, however, they do hold data in a similar way, in fact, most Collections are wrappers around Arrays.

An Array is an Object (Reference Type) that holds value types (Primitive types or references to Objects) as an indexed set.

- The Array index starts at 0.
- The number of elements the Array will hold must be declared when the array is instantiated
- Once Instantiated the size of the array cannot be changed, so if during instantiation it is set to hold 5 elements, then that Array will always hold 5 elements.
- Must be instantiated with the new keyword

Syntax

<data type>**[]** <variable name> **= new** <data type>**[**<number of elements>**]**;

<data type>	The data type of values the array will hold. This can be any primitive type (int, boolean, double, byte, etc.) or Object Type (class, interface, or abstract)
<variable name>	A name for the variable that will hold the array was created.
<number of elements>	The number of elements the array will hold

Examples:

`int[] intArray = new int[5];`	This array will hold a set of 5 ints
`String[] stringArray = new String[10];`	This array will hold a set of 10 Strings
`House[] houses = new House[4];`	This array will hold a set of 4 Houses
`DeliveryDriver[] drivers = new DeliveryDriver[6];`	This array will hold a set of 6 DeliveryDrivers

There is also a shortened syntax that allows for elements to be assigned to the array on instantiation, using this syntax the array will be sized to the number of elements provided.

Examples:

`int[] numbers = new int[] { 1, 2, 3, 4, 5 };`

This will create a new Array of ints that has 5 elements that will contain 1, 2, 3, 4, 5

`String[] names = new String[] { 'Sally', 'Jane', 'Joe', 'Sam' };`

This will create a new Array of Strings that has 4 elements that will contain Sally, Jane, Joe, and Sam

`DeliveryDriver[] drivers = new DeliveryDriver[] { new FexEd(), new SPU(), new PostalService() };`

This will create a new Array of DeliveryDrivers that has 3 elements of type DeliveryDriver. In this case DeiveryDriver is an interface, so the elements must be implementation classes that implement the DeliveryDriver interface, so it will contain objects of type FexEd, SPU, and PostalService.

Structure

Internally Arrays hold the values in indexed positions as a set that is associated with the array. Arrays are 0 indexed, but the number of elements is a count, so it begins with 1. So if we define an Array with 4 elements, it will index the elements as 0, 1, 2, 3. If no values are given, then the array will set the values to that data types default (0 for numeric primitives, false for boolean, and null for objects. If the shorthand syntax is used to assign values to the array on instantiation, then those values will be populated as values in the order they are given.

Examples

Array Definition	Internal Structure
`int[] intArray = new int[4];`	**intArray** **Index** / **Initial Value** 0 / 0 1 / 0 2 / 0 3 / 0
`House[] houses = new House[2];`	**houses** **Index** / **Initial Value** 0 / null 1 / null
	trueOrFalse

```
boolean[] trueOrFalse = new boolean[3];
```

Index	Initial Value
0	false
1	false
2	false

```
int[] numbers = new int[] { 1, 2, 3, 4, 5 };
```

numbers

Index	Initial Value
0	1
1	2
2	3
3	4
4	5

```
String[] names = new String[] { 'Sally', 'Jane', 'Joe', 'Sam' };
```

names

Index	Initial Value
0	Sally
1	Jane
2	Joe
3	Sam

Access Array Elements

Once the array has been instantiated the elements can be accessed by their index by referring to it in [ ] after the variable name that the array is assigned. When accessed in this way, the array with index can be used just like a variable of that type.

Examples

```
String[] names = new String[] { 'Sally', 'Jane', 'Joe', 'Sam' };
```

Index	**Initial Value**
0	Sally
1	**Jane**
2	Joe
3	**Sam**

names**[1]** refers to the String with a value of **Jane**

names**[3]** refers to the String with a value of **Sam**

Since the array values are a String, referring to it this way allows the value at that index to be used like a String:

if (names**[1]**.equals("Jane")) → result is TRUE

names**[3]**.subString(1) → returns "am"

int[] numbers = new int[] { 1, 2, 3, 4, 5 };

Index	**Initial Value**
0	**1**
1	2
2	3
3	**4**
4	**5**

numbers**[0]** refers to the int with the value **1**

numbers**[3]** refers to the int with the value **4**

numbers**[4]** refers to the int with the value **5**

Since the array values are a int, referring to it this way allows the value at that index to be used like an int:

if (numbers**[0]** >= 1) ← result is TRUE

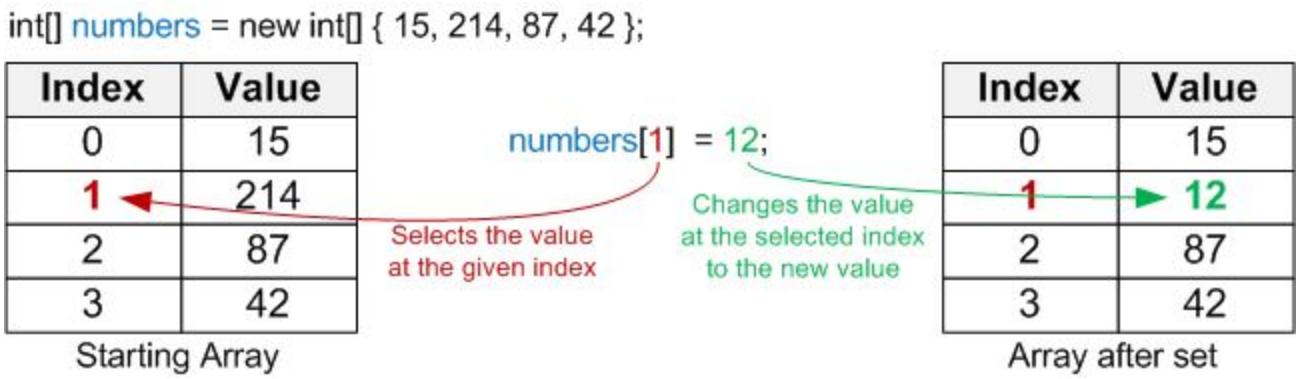
int y = numbers**[4]** * 2; ← results in y == 10

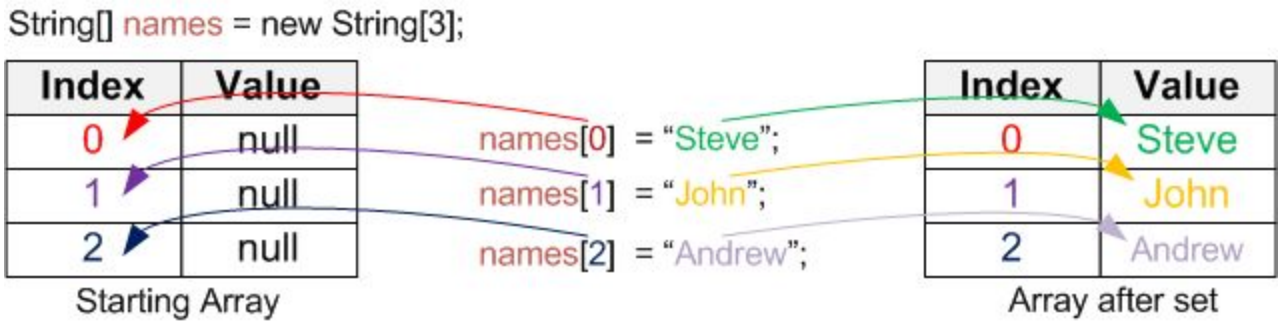
Setting Array Values

Array values can be set by accessing them using the same variable_name[index] syntax used to access them. Since this syntax treats the value at this element as a variable, the assignment operator (=) can be used.

Examples

```
intArray[3] = 2;          ← sets the value at the 3rd index to 2
stringArray[4] = "Jill";  ← sets the value at the 4th index to "Jill"
```





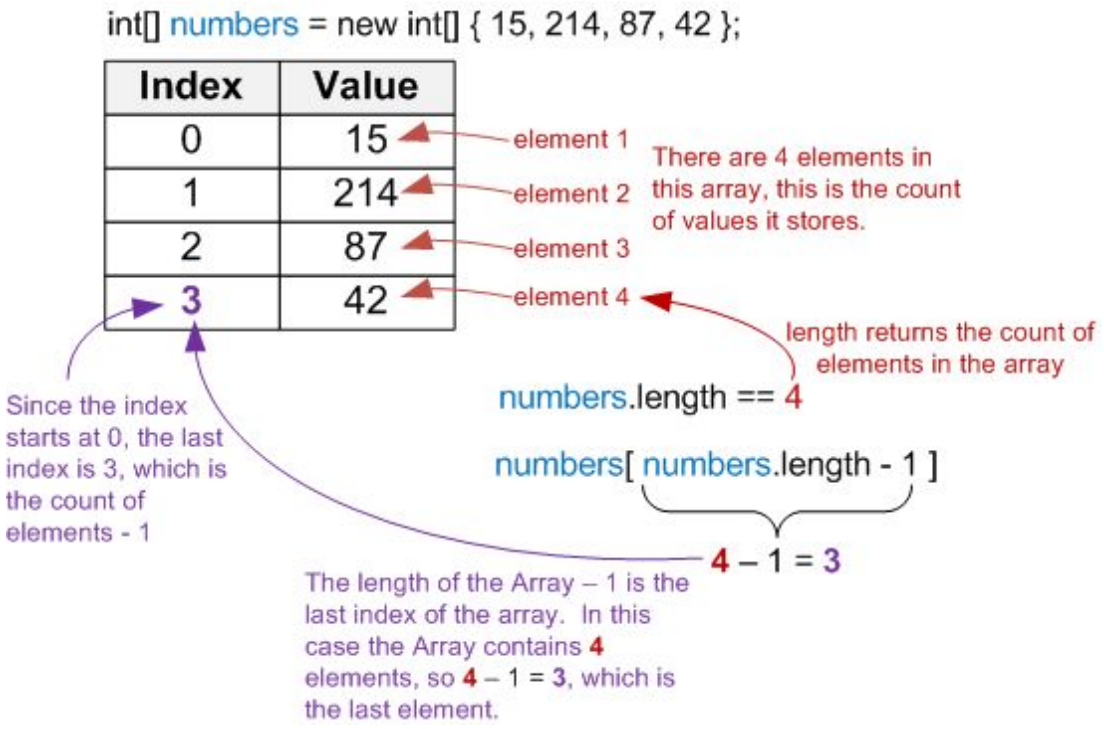
Array Properties

Arrays have a length property that can be accessed to determine the number of elements the array contains. Since arrays set default values for all fields, the number returned by length returns the size of the array set on creation. Array.length returns the count of elements in the array and not the last index. The count starts at 1, while the index starts at 0, therefore, the last element of the array will always be at Array.length - 1.

Examples

```
int[] intArray = new int[4];
int.length  ← returns 4
```

```
String[] names = new String[] { "Jim", "Steve", "John" }
names.length  ← returns 3
```



Arrays with Loops

Arrays can be used with all types of loops. The most common loop to use with an Array is a standard for loop. Since the incrementer in the for loop traditionally starts at 0, it can be used to access each element of the array as it increments.

For loop Example

```
int[] intArray = new int[] { 22, 42, 78 };

for (int i = 0 ; i < intArray.length ; i++ ) {

        intArray[i] = intArray[i] * 2;
}
```

int **i** = 0	The incrementer is started at 0, which will be the first index of the Array
i < intArray.**length**	The loop will continue as long as incrementer is less than the length of the array. Less Than (<) is used since the last index of the array is at Array.length - 1. i <= intArray.length - 1 could also be used.
intArray[**i**]	The array values are accessed using the incrementer as the index. Since the index starts at 0 and then increases by 1 each loop, this will access each element of the array in order. In this case the array

	value at the index i is being set to a new value.
intArray[i]	The array value at the same index is accessed again using i, but in this case it is getting the original value at that index, so it can use it in the calculation.
i++	At the end of each loop the incrementer is increased by 1, which will allow access to the next index of the array.

Explanation

Loop iteration	Value of i	Value of intArray[i] at start of iteration	Value of intArray[i] at end of iteration
1	0 →	intArray[0] is 22	intArray[0] = 44 (22 * 2)
2	1 →	intArray[1] is 42	intArray[1] = 84 (42 * 2)
3	2 →	intArray[2] is 78	intArray[2] = 156 (78 * 2)
4	3 →	Array.length is 3, so i (3) < intArray.length (3) is not true and the loop ends.	

For Each loop Example

```
String[] namesArray = new String[] { 'Sally', 'Jane', 'Joe', 'Sam' };

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

namesArray	The variable that holds the array to loop through
String name	A variable to hold the current value of the array. The type must be the same as the Array.
name	The variable name can be used in the loop body. Each iteration it will contain the next item from the array.

Explanation

{ ‘Sally’, ‘Jane’, ‘Joe’, ‘Sam’ };

Loop iteration	**Value of** name
1	name → Sally
2	name → Jane
3	name → Joe
4	name → Sam
5	No more items in the array, so the loop ends