Learning Objectives: Array Basics

- **Create an array using both the *initializer list* and *new* methods**
- **Access and modify array elements**
- **Iterate through arrays using both a regular `for` loop and an *enhanced* `for` loop**
- **Determine array output**

Creating an Array

What Is an Array?

An **array** is a data structure that stores a collection of data such as ints, doubles, Strings, etc. This data is often referred to as the array's **elements**. Being able to store elements into an array helps reduce the amount of time needed to declare and initialize variables. For example, if you wanted to store the names of all family members in your household, you would typically have to declare and initialize String variables and values for each family member. Copy the code below into the text editor on the left and then click the `TRY IT` button to see the output. You can also click on the ++Code Visualizer++ link underneath to see how the program runs behind the scenes.

```
String a = "Alan";
String b = "Bob";
String c = "Carol";
String d = "David";
String e = "Ellen";

System.out.println(a);
```

Code Visualizer

challenge

What happens if you:

- Change the `a` in `System.out.println(a)` to `b`, `c`, `d`, or `e`?

Code Visualizer

Array Creation

To avoid the repetitive task of declaring and initializing multiple variables, you can declare an array and directly assign values or elements into that array like below. This technique is referred to as the **initializer list** method.

```
String[] names = {"Alan", "Bob", "Carol", "David", "Ellen"};
```

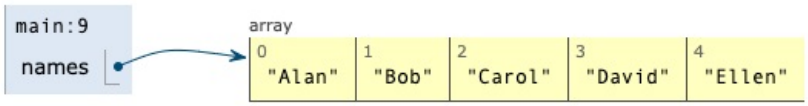
Code Visualizer

“Initializer List” Method Syntax:

- Specify the data type that the array will store (i.e. `String`) followed by empty brackets `[]`.
- Declare the variable name for the array (i.e. `names`) followed by the assignment symbol `=`.
- Elements assigned to the array are separated by commas `,` and enclosed within curly braces `{}`.

▼ **Additional information**

If you used the Code Visualizer, you’ll notice that the array variable `names` refers to all of the elements as a collection. An array is considered to be an **object** that bundles all of the data that it holds.



Note that the first array slot, or **index**, is always `0` so `Alan` is located at index `0` instead of `1`.

Alternatively, you can create an array using the **new** method in which you will need to declare and specify the array variable and length before you can assign elements to the array.

```
String[] names = new String[5];
```

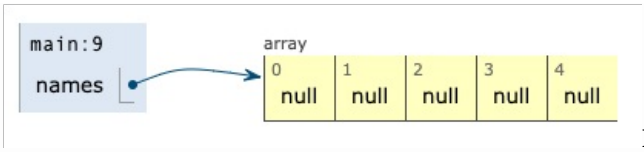
Code Visualizer

“New” Method Syntax

- Specify the data type that the array will store (i.e. `String`) followed by empty brackets `[]`.
- Declare the variable name for the array (i.e. `names`) followed by the assignment symbol `=`.
- Declare the keyword `new` followed by the data type (i.e. `String`) and number of elements in brackets (i.e. `[5]`).

▼ **Additional information**

If you used the Code Visualizer, you’ll notice that the array variable `names` refers to all of the elements as a collection. `null` appears in all of the array slots because no elements have been assigned to them yet.

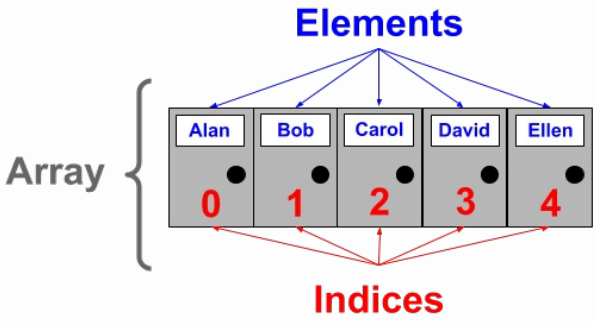


Note that the first array slot, or **index**, is always `0`.

Array Details

Both the **initializer list** and **new** methods mentioned above will store five elements in the array called `names`. However, the *new* method automatically initializes each element to a default value of `null` while the *initializer list* method does not. Note that array slots are formally called **indices** and each **index** can carry just one type of data. For example, storing an `int` of `5` into a `String` array will result in a error.

P.O. Boxes at the postal office is symbolically similar to arrays. Each row of P.O. Boxes is like an array, except each box can only store *one* item (element) *and* each item within that row must be of the same *type* (i.e. Strings). The position at which each box is located is its index.



.guides/img/ArrayElementsIndices

Accessing an Array

Array Access

To access and print array elements, you need to know their position. The position at which an element is stored is called its **index**. For example, `numbers[0]` refers to the first element in the array called `numbers`. Array indices always start at `0` and increment by 1 with each element that comes next. Due to this, `numbers[4]` refers to the *fifth* element in the array, *not* the fourth.

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

System.out.println(numbers[0]);
```

Code Visualizer

challenge

What happens if you:

- Change `numbers[0]` in the code above to `numbers[2]`?
- Change `numbers[0]` in the code above to `numbers[3]`?
- Change `numbers[0]` in the code above to `numbers`?

Code Visualizer

important

IMPORTANT

You may have noticed that printing the `numbers` array without specifying an index resulted in an output that starts with `[I@`... This occurs because printing an array actually prints its memory location, not its elements. You'll learn how to print all elements in an array without having to specify all of their indices on a later page.

Default “New” Elements

When using the *new* method to create an array, there are default values that populate as elements inside of the array, depending on the array type. For example, `String[] words = new String[5]` will result in *five* `null` elements and `int[] numbers = new int[5]` will populate *five* elements of `0`s within the array. Below is a table showing the default values of different array types when the “new” method is used.

Data Type	Default Value
String	null
int	0
double	0.0
boolean	false

```
double[] decimals = new double[2];
boolean[] bools = new boolean[2];

System.out.println(decimals[0]);
System.out.println(bools[0]);
```

Code Visualizer

challenge

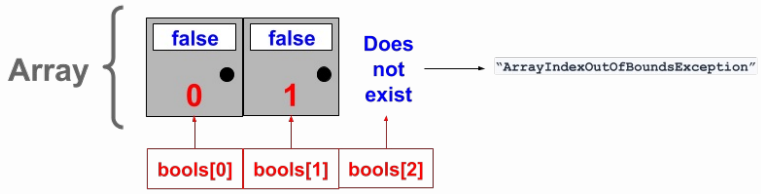
What happens if you:

- Change `decimals[0]` in the code above to `decimals[1]`?
- Change `bools[0]` in the code above to `bools[1]`?
- Change `decimals[0]` in the code above to `decimals[2]`?
- Change `bools[0]` in the code above to `bools[2]`?

Code Visualizer

IndexOutOfBounds Error

A common error that occurs is called the `ArrayIndexOutOfBoundsException` error. This happens when you try to access or print an element at an index that does not exist within the array. In the example above, `decimals[2]` and `bools[2]` both resulted in an `ArrayIndexOutOfBoundsException` because neither array has an index of `2`. Both arrays have only indices `[0]` and `[1]` to hold their elements.



.guides/img/ArrayException

Modifying an Array

Array Modification

To modify an element within an array, simply find the index at which that element is stored and assign a new value to it.

```
int[] grades = {85, 95, 48, 100, 92};
System.out.println(grades[2]);

grades[2] = 88; //88 will replace 48 at index 2
System.out.println(grades[2]);
```

Code Visualizer

challenge

What happens if you:

- Change `int[] grades = {85, 95, 48, 100, 92};` in the code above to `int[] grades = new int[5];`?
- Change all `System.out.println(grades[2]);` in the code above to `System.out.println(grades[3]);`?
- Change `grades[2] = 88;` in the code above to `grades[3] = 100;`?

Code Visualizer

Modifying Multiple Arrays

You can create and modify as many arrays as you'd like. For example, you can create an array to store your family members and another array to store their age.

```
String[] family = {"Dad", "Mom", "Brother", "Sister"};
int[] age = new int[4];

System.out.println(family[0] + " " + age[0]);
System.out.println(family[1] + " " + age[1]);
System.out.println(family[2] + " " + age[2]);
System.out.println(family[3] + " " + age[3]);
```

Code Visualizer

challenge

What happens if you:

- Add `age[0] = 50;` directly below the line `int[] age = new int[4];`?
- Add `age[1] = 45;` below the line `int[] age = new int[4];` but before the print statements?
- Add `age[2] = 25;` below the line `int[] age = new int[4];` but before the print statements?
- Add `age[3] = 20;` below the line `int[] age = new int[4];` but before the print statements?
- Change `"Sister"` within the String array to `"Brother2"`?

Code Visualizer

important

IMPORTANT

Since the integer array above was created using the *new* method, `0` was populated as elements within the array at first. Then by setting the array indices to specific values, you were able to modify the array to include the appropriate age for each family member.

Iterating an Array

Array Iteration

Though we can add many elements to our array, printing each of them can get quite tedious. For example, if we have 10 names of friends in our array, we would need to specify each of their array index to print them.

```
String[] friends = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

System.out.println(friends[0]);
System.out.println(friends[1]);
System.out.println(friends[2]);
System.out.println(friends[3]);
System.out.println(friends[4]);
System.out.println(friends[5]);
System.out.println(friends[6]);
System.out.println(friends[7]);
System.out.println(friends[8]);
System.out.println(friends[9]);
```

Code Visualizer

Luckily, we can use loops which we had learned previously to help us with this process. To print out all of our friends' names without repeating the print statement ten times, we can use a `for` loop to iterate `10` times.

```
String[] friends = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

for (int i = 0; i < 10; i++) {
  System.out.println(friends[i]);
}
```

Code Visualizer

challenge

What happens if you:

- Change `System.out.println(friends[i]);` in the code above to `System.out.println(friends[0]);`?
- Change `System.out.println(friends[i]);` in the code above to `System.out.println(friends[10]);`?

Code Visualizer

important

IMPORTANT

Did you notice that the print statement above includes `i` as the index for `friends`? We do this because `i` will take on the values specified by the `for` loop. The loop starts at `0` and increments by 1 until it reaches `9` (not including 10). Thus, `friends[0]` will print, then `friends[1]`, so on and so forth until `friends[9]` is printed. Then the loop ends.

Array Length

To make the iteration process easier, we can use an instance variable called `length` to determine how many elements are in our array. To use `length`, just call it by adding a period `.` after our array followed by `length`. For example, `friends.length` will tell us how many elements are in our `friends` array. The advantage of using `length` is that we can initialize additional elements in our array without having to keep track of how many elements are already inside.

```
String[] friends = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

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

Code Visualizer

challenge

What happens if you:

- add `"Kate"` as an element to the array right after `"Jen"`?
- remove `"Alan"` and `"Bob"` from the array?

Code Visualizer

Notice how `friends.length` continues to keep track of how many elements are in our array even though we've made several changes.

Enhanced For-Loop

Using an Enhanced For-Loop

There is a special type of `for` loop that can be used with arrays called an **enhanced for loop**. An enhanced `for` loop, also known as a **for each loop**, can be used to iterate through array elements without having to refer to any array indices. To use an enhanced `for` loop, you need the following:
* The keyword `for` followed by parentheses `()`.
* A **typed** iterating variable followed by colon `:` followed by the array name.
* **Note** that the iterating variable must be of the *same* type as the array.
* Any commands that repeat within curly braces `{}`.
* **Note** that when using an enhanced `for` loop, you can print the iterating variable itself without using brackets `[]`.

```
String[] friends = {"Alan", "Bob", "Carol", "David", "Ellen",
                    "Fred", "Grace", "Henry", "Ian", "Jen"};

for (String i : friends) {
  System.out.println(i);
}
```

Code Visualizer

challenge

What happens if you:

- change `System.out.println(i);` in the code above to `System.out.println(friends[i]);`?
- change `String i` in the code above to `int i`?

Code Visualizer

important

IMPORTANT

One of the main differences between a regular `for` loop and an enhanced `for` loop is that an enhanced `for` loop does not refer to any index or position of the elements in the array. Thus, if you need to access or modify array elements, you **cannot** use an enhanced `for` loop. In addition, you **cannot** use an enhanced `for` loop to iterate through a *part* of the array. Think of an enhanced `for` loop as an *all-or-nothing* loop that just prints all of the array elements or nothing at all. Also note that the iterating variable type **must match** the array type. For example, you cannot use `for (int i : friends)` since `friends` is a String array and `i` is an integer variable. Use `for (String i : friends)` instead.

Helpful Array Algorithms

Array Algorithms

In addition to being used with loops, arrays can also be used with conditionals to help with tasks such as searching for a particular element, finding a minimum or maximum element, or printing elements in reverse order.

Searching for a Particular Element

```
String[] cars = {"Corolla", "Camry", "Prius", "RAV4",
        "Highlander"};
String Camry = "A Camry is not available."; //default String
        value


for (String s : cars) { //enhanced for loop
  if (s.equals("Camry")) { //if "Camry" is in array
    Camry = "A Camry is available."; //variable changes if
        "Camry" exists
  }
}

System.out.println(Camry); //print whether Camry exists or not
```

Code Visualizer

challenge

What happens if you:

- delete `"Camry"` from the `cars` array?
- try to modify the code above so that the algorithm will look for `Prius` in the array and will print `A Prius is available.` if `Prius` is an element and `A Prius is not available.` if it is not an element.

Code Visualizer

▼ **Sample Solution**

```
String[] cars = {"Corolla", "Camry", "Prius", "RAV4",
        "Highlander"};
String Prius = "A Prius is not available.";

for (String s : cars) {
  if (s.equals("Prius")) {
    Prius = "A Prius is available.";
  }
}

System.out.println(Prius);
```

Finding a Minimum or Maximum Value

```
int[] grades = {72, 84, 63, 55, 98};
int min = grades[0]; //set min to the first element in the array

for (int i : grades) { //enhanced for loop
  if (i < min) { //if element is less than min
    min = i; //set min to element that is less
  }
}
//elements are not modified so enhanced for loop can be used

System.out.println("The lowest grade is " + min); //print lowest
        element
```

Code Visualizer

challenge

What happens if you:

- replace 72 in the int array with 42?
- try to modify the code so that the algorithm will look for the **maximum** element instead?

Code Visualizer

▼ **Sample Solution**

```java
int[] grades = {72, 84, 63, 55, 98};
int max = grades[0];

for (int i : grades) {
  if (i > max) {
    max = i;
  }
}

System.out.println("The highest grade is " + max);
```

Printing Elements in Reverse Order

```java
String[] letters = {"A", "B", "C", "D", "E"};

//start at index 4, then decrement by 1 until i < 0, then stop
for (int i = letters.length - 1; i >= 0; i--) {
  System.out.println(letters[i]);
}

//regular for loop needed to access each element index
```

Code Visualizer