

# Introduction to Arrays

by Sophia



## WHAT'S COVERED

In this lesson, you will learn about using a data structure called an array, to store and access multiple data values of the same type. You will also learn array access. Specifically, this lesson covers:

## Table of Contents

- [1. Arrays](#)
- [2. Array Access](#)

## 1. Arrays

When working with code, there are times that you will be working with one element at a time. Using a name or a single price are examples of the use of an array. There are other instances where there is a need to work with larger sets of data. An **array** is a special type of variable that represents multiple values that are all of the same data type. Most programming languages have a structure that allows users to store multiple values of the same type, in addition to storing single values. To better understand the code used when working with arrays, it's important to understand what an array is, how it works, and why one would want to use an array in code.



### CONCEPT TO KNOW

An **array** is a named collection of contiguous storage locations. These are storage locations that are next to each other. They contain data items of the same type. Each of the items (values) is called an **element**. The elements are arranged in a linear fashion. Each element has an **index**, which is also called a subscript. It is an integer that specifies its location in the array. The first item in the array has the index 0, and the index of the last element is one less than the length of the array.

Java arrays can be single or multi-dimensional. The first part of this tutorial will focus on single-dimensional arrays. Multi-dimensional arrays will be covered later in this tutorial. When declaring an array, the declaration begins with the data type. When declaring an array, the data type is immediately followed by a pair of square brackets.

The type and square brackets are followed by the name (identifier), which is similar to how plain variables are declared:

## ➤ EXAMPLE

```
int[] scores;
```

The statement above declares an array of int values but does not provide a size. The array has to have its size set before data can be read from or written to it. Specifying the initial values for the array is one way to set the size.

The initial values are specified in a comma-separated list in curly brackets:

## ➤ EXAMPLE

```
int[] scores = {100, 99, 98, 97, 96};
```

This array can be pictured like this:

Scores					
Index	0	1	2	3	4
Value (int)	100	99	98	97	96

An array of five String values can be declared like this:

## ➤ EXAMPLE

```
String cheeseChoice = {"Cheddar", "Swiss", "Gouda", "American", "Mozzarella"};
```

This array can be illustrated like this:

cheeseChoice					
Index	0	1	2	3	4
Value (String)	"Cheddar"	"Swiss"	"Gouda"	"American"	"Mozzarella"

Arrays can also be declared without specifying the actual values. To declare an empty array with a certain size, use the new keyword followed by the data type and size.

This is done in square brackets as demonstrated in this code:

## ➤ EXAMPLE

```
int[] scores = new int[5];
```

An empty array that can hold five String values can be declared like this:

## ➤ EXAMPLE

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



## BIG IDEA

When working with arrays, it is important to keep in mind that once the type and size of an array have been declared, these cannot be changed, as the type and size are immutable. We will discuss a workaround that allows copying the contents of an array to a larger array later in the challenge.

When experimenting by defining arrays in Java, you will find that it is possible to declare an array by placing the square brackets after the name of the array rather than after the data type for the array.

The following declaration works, but it is not the preferred approach:

### → EXAMPLE

```
int scores[] = new int[5];
```

This style of array declaration, with the square brackets after the name rather than the data type, is found in programming languages such as C and C++. While this syntax may have once made early versions of Java seem more familiar to C and C++ programmers, it is no longer recommended. Standard practice in Java places the square brackets after the data type, as shown earlier in this section.



## THINK ABOUT IT

Why do we use arrays? Think of a real-world example of an array as a parking lot. Look at the entire parking lot as a whole and think of it as a single object. Inside of the parking lot there are parking spots that are uniquely identified by a number. There may be parking spots that have a car, a truck, a motorcycle, or may even be empty. Within a list, the first spot is identified as a 0. This means that if there are 10 elements in a list, the index values go from 0 to 9. If there are five spots, the index values of the spots go from 0 to 4.

Index values in a list look like this:

### → EXAMPLE

```
String[] parkingLot = {"motorcycle", "", "truck", "car", "car"};
```

This code can be illustrated as in the following table:

parkingLot List					
Index (spot #)	0	1	2	3	4
Value	"motorcycle"	" "	"truck"	"car"	"car"



## TERMS TO KNOW

### Array

An array is a named collection of contiguous storage locations—storage locations that are next to each other—that contain data items of the same type. An array also has a fixed size.

## Element

An element is one of the values in an array.

## Index

The index (also known as the subscript or position) is an integer value that indicates an element in an array.

---

# 2. Array Access

As discussed previously, array elements are indexed. This means that the first element in the array has an index of 0. Then, the second element has an index of 1. This structure continues until all elements are indexed. Each element in the array is ordered using this index.

Each element in the array will follow this syntax:

### ➞ EXAMPLE

```
array[x]
```

In this example below, the array is replaced with the name of the array that is to be accessed, and the x is replaced with the desired index (position) number. The first element is always 0 and not 1. This means that the elements in the array have a defined order and that order will not change unless we modify the order. You can assign array values using the name of the array and the assignment operator, or the = sign.

A value can also be assigned to an individual element in the array using the assignment operator ( = ) and the name of the array and the index, as seen below:

### ➞ EXAMPLE

```
int[] numbers = new int[3];  
numbers[0] = 10;  
numbers[1] = 20;  
numbers[2] = 30;
```

After these statements, the numbers array contains the integer values of 10, 20, and 30.



### CONCEPT TO KNOW

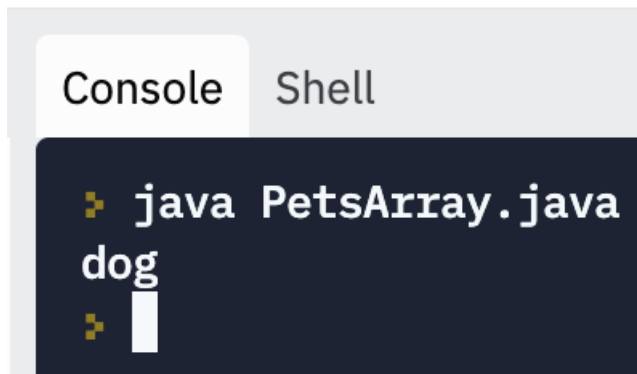
Arrays are **mutable**. This means that the values can be changed in the elements. The syntax for accessing the elements of an array uses the bracket operator. The expression inside the brackets specifies the index of the element to be accessed. This index is the position number. Remember that the indices start at 0.

Here is an example of this expression:

```
class PetsArray {  
    public static void main(String[] args) {  
        String[] pets = {"dog", "cat", "fish"};
```

```
System.out.println(pets[0]);  
}  
}
```

Running this code produces this result:



**Directions:** Try it out yourself. Output the element in the array that has the value “fish”:

What would the index need to be set to? Click the plus (expand) icon on the right to see if you are correct.



Were you correct? The index would need to be 2 to retrieve “fish” from the array.

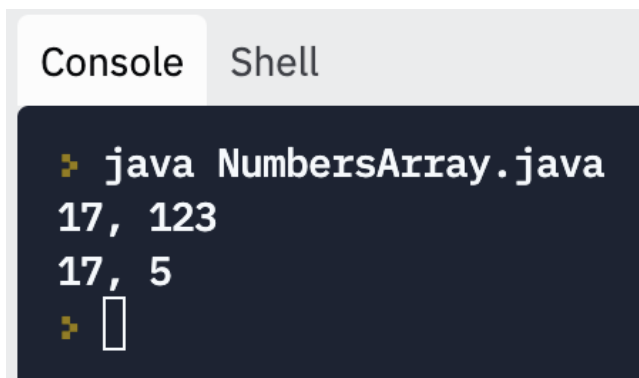


Arrays are mutable because the values of elements are changed. Or, they are assigned a new value to an element, as long as the data type is correct. When the bracket operator appears on the left side of the assignment operator, it identifies the element of the array to which the value will be assigned.

Here is an example of an array and the element to which a value will be assigned:

```
class NumbersArray {  
    public static void main(String[] args) {  
        int[] numbers = {17, 123};  
        System.out.println(numbers[0] + ", " + numbers[1]);  
        numbers[1] = 5;  
        System.out.println(numbers[0] + ", " + numbers[1]);  
    }  
}
```

Here is what the output should look like:



```
Console Shell
❏ java NumbersArray.java
17, 123
17, 5
❏ []
```

### THINK ABOUT IT

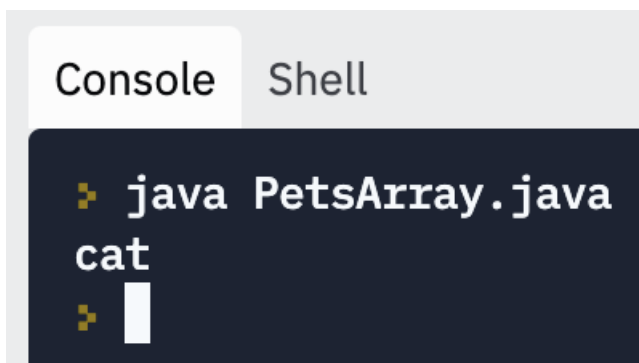
As this small program shows, the values of an array can have their values changed. The key points are to choose the correct index for the element to be updated and to make sure that the new value is of the correct data type. The element in index 1 (the second number) of the numbers array, which used to be 123, is now 5.

An array has a relationship between indices (index) and elements. This relationship is called a mapping. In this scenario, each index “maps to” one of the elements. Any integer expression can be used as an index.

Here is an example of an unusual (but still functional) integer expression:

```
class PetsArray {
    public static void main(String[] args) {
        String[] pets = {"dog", "cat", "fish"};
        System.out.println(pets[5-4]);
    }
}
```

The output should look like this:



```
Console Shell
❏ java PetsArray.java
cat
❏
```

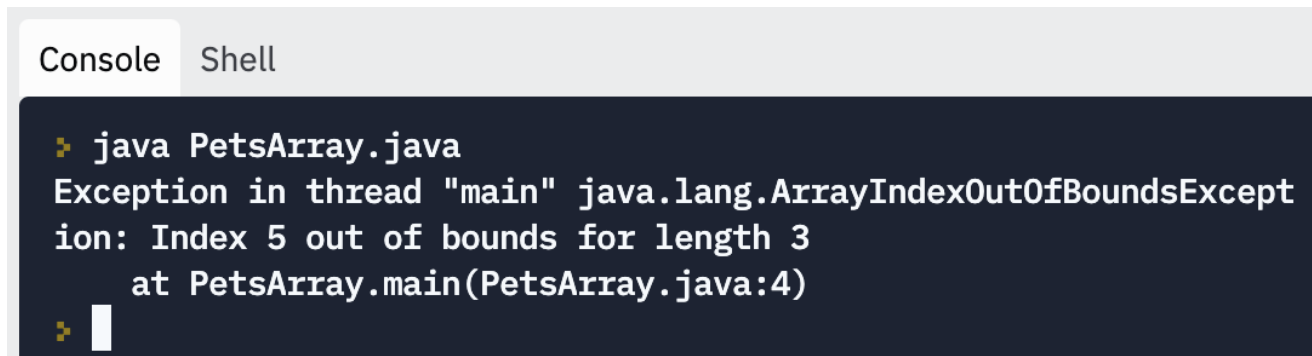
In the code above, 5 minus 4 is the same as if we had 1, so it would print out index 1, or cat.

Attempting to print an element that does not exist will cause an `IndexError` like this one:

```
class PetsArray {
    public static void main(String[] args) {
        String[] pets = {"dog", "cat", "fish"};
        System.out.println(pets[5]);
    }
}
```

```
}
```

The output should look like this:

A screenshot of a Java IDE's console window. The window has two tabs: 'Console' and 'Shell'. The 'Console' tab is active, showing the output of a Java program. The output text is: 

```
java PetsArray.java
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
    at PetsArray.main(PetsArray.java:4)
```

In the example above, the index values extend from 0 to 2, so index 5 produces an exception.

In a previous challenge, you learned how Java uses try and catch blocks to deal with exceptions. A try block could be used here when accessing the array.

A corresponding catch block can be used to handle the `ArrayIndexOutOfBoundsException` like the one below:

```
import java.util.Scanner;

class PetsArray {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String[] petList = {"dog", "cat", "fish"};
        System.out.println("Select a pet: ");
        System.out.println("1. " + petList[0]);
        System.out.println("2. " + petList[1]);
        System.out.println("3. " + petList[2]);
        System.out.print("Enter selection #: ");
        int choice = input.nextInt();
        // Subtract 1 from choice
        choice--;
        // try to access element in array
        try {
            System.out.println("You selected a " + petList[choice]);
        }
        catch(ArrayIndexOutOfBoundsException ex) {
            System.out.println("Not a valid selection.");
        }
    }
}
```

Running this program and entering an out-of-range exception looks like this:

```
> java PetsArray.java
Select a pet:
1. dog
2. cat
3. fish
Enter selection #: 4
Not a valid selection.
> 
```

As discussed in the previous tutorial about handling errors, when using a Scanner to read input, users can use try and catch to react to invalid input. An example of this situation includes incidents where the entry isn't a number.

The full program then looks like this:

```
import java.util.Scanner;

class PetsArray {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        String[] petList = {"dog", "cat", "fish"};
        System.out.println("Select a pet: ");
        System.out.println("1. " + petList[0]);
        System.out.println("2. " + petList[1]);
        System.out.println("3. " + petList[2]);
        System.out.print("Enter selection #: ");
        // Declare the variable choice before try block so it can
        // be used later in the program.
        int choice = 0;
        try {
            choice = input.nextInt();
        }
        catch(Exception ex) { // If user doesn't enter a number
            System.out.println("That's not a number.");
        }
        // Subtract 1 from choice
        choice--;
        // try to access element in array
        try {
```

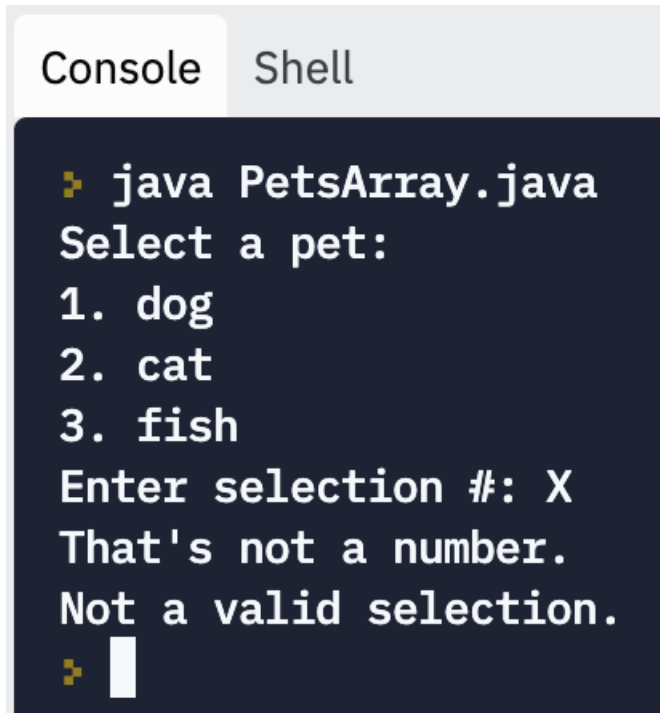


```

        System.out.println("You selected a " + petList[choice]);
    }
    catch(ArrayIndexOutOfBoundsException ex) { // Number out of range
        System.out.println("Not a valid selection.");
    }
}
}

```

If this program is run and a letter is entered rather than a number, the program produces this result:



```

Console  Shell
❏ java PetsArray.java
Select a pet:
1. dog
2. cat
3. fish
Enter selection #: X
That's not a number.
Not a valid selection.
❏

```



#### TERM TO KNOW

#### Mutable

Mutable means that we're able to change the value of items in an array.



#### SUMMARY

In this lesson, you learned about **arrays**. You discovered that an array is the simplest data structure in Java that can store multiple values. These values are called elements in a single variable. You also learned how to create an array, and that **accessing an array** element can be done using an index; remember that array indexes start at 0 for the first element. Finally, you learned that arrays are mutable and that you can change the order of elements in an array or reassign an element in an array.

Source: This content and supplemental material has been adapted from Java, Java, Java: Object-Oriented Problem Solving. Source [cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf](https://cs.trincoll.edu/~ram/jjj/jjj-os-20170625.pdf)

It has also been adapted from "Python for Everybody" By Dr. Charles R. Severance. Source [py4e.com/html3/](https://py4e.com/html3/)

**Array**

An array is a named collection of contiguous storage locations—storage locations that are next to each other—that contain data items of the same type. An array also has a fixed size.

**Element**

An element is one of the values in an array.

**Index**

The index (also known as the subscript or position) is an integer value that indicates an element in an array.

**Mutable**

Mutable means that we're able to change the value of items in an array.