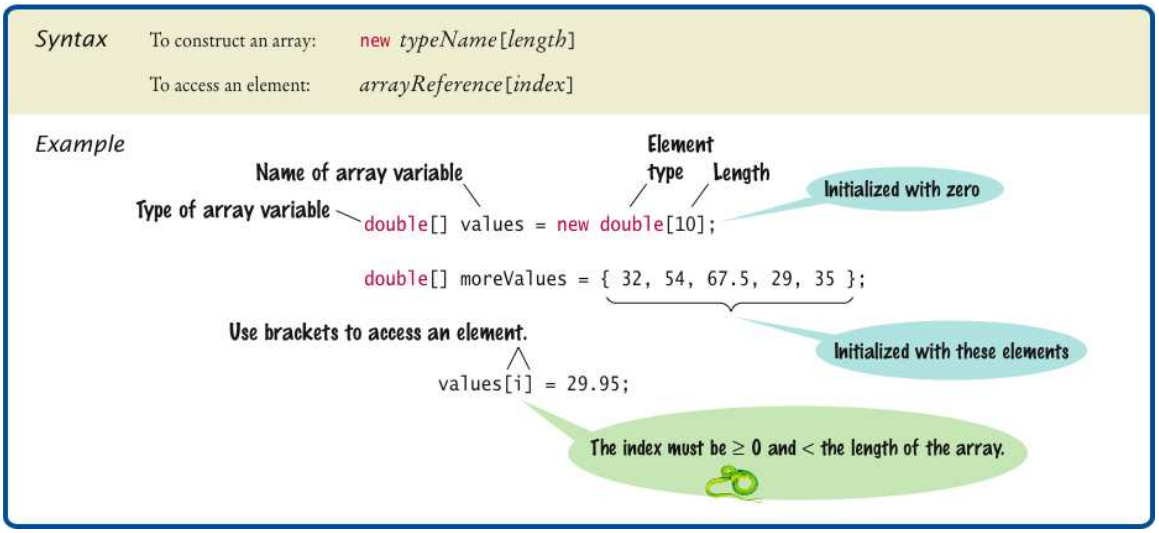
# I. Arrays

week 12/13

- an array is a data structure which stores/holds values of the **same type**
- the individual units of an array are called its "elements"
- arrays must be given a
  - name – to identify the array itself
  - type – what type of data will the array store
  - size (or length) – how many data values can the array store/hold
  - index – what specific element is being referenced
- **Note: array index range will always start at 0 and will end at size – 1**
- declaration syntax:

```
type  [] var_name = new type[length];          // where length must be an int type
type var_name  [] = new type[length];
```




- Note: when you declare an array of primitive types, numeric types will be set to zero, and boolean types will be set to a zero-like value (i.e. false). Reference types will also be set to something zero-like (i.e. null).

| Table 7.1. Declaring Arrays | |
|---|---|
| `int[] numbers = new int[10];` | An array of ten integers. All elements are initialized with zero. |
| `final int NUMBERS_LENGTH = 10;`<br>`int[] numbers = new int[NUMBERS_LENGTH];` | It is a good idea to use a named constant instead of a "magic number". |
| `int valuesLength = in.nextInt();`<br>`double[] values = new double[valuesLength];` | The length need not be a constant. |
| `int[] squares = { 0, 1, 4, 9, 16 };` | An array of five integers, with initial values. |
| `String[] names = new String[3];` | An array of three string references, all initially `null`. |
| `String[] friends = { "Emily", "Bob", "Cindy" };` | Another array of three strings. |
| `double[] values = new int[10]` | Error: You cannot initialize a `double[]` variable with an array of type `int[]`. |

# Essential skills for working with arrays

This chapter begins by showing how to use the Java language to create and work with arrays.

## How to create an array

An *array* is an object that contains one or more items called *elements*. Each element within an array can be a primitive type such as an int value or a reference type such as a String object or a Product object. As you'll see later in this chapter, an array itself is a reference type.

In Java, all of the elements in an array must be of the same type. As a result, an array of the int type can only contain int values, and an array of the String type can only contain String objects. However, an array can contain elements that are derived from the array's base type. As a result, if you declare an array of the Product type, the array can contain any object that inherits the Product class. You'll learn more about how this works in the next chapter.

The *length* (or *size*) of an array indicates the number of elements it contains. In Java, arrays have a fixed length. As a result, once you create an array, you can't change its length. If you need to change the length of an array, you should consider using a collection instead of an array. For example, you may want to use the ArrayList class described in chapter 14 instead of an array. Then, the collection automatically adjusts its length to fit the array.

Figure 10-1 shows several ways to create an array. To start, you must declare a variable that refers to the array object. Then, you create an array object and assign it to the variable. To create a new array object, you code the new keyword, followed by the data type, followed a pair of brackets. Within the brackets, you code the length of the array.

You can use separate statements to declare the array variable and create the array object as shown in the first example. Or, you can declare the variable and instantiate the array object in a single statement as shown in the second example. Both of these examples create an array that holds four double values.

When you declare the array variable, you use an empty set of brackets to indicate that the variable is an array. Most programmers prefer to code the empty brackets after the data type to indicate that the array is an array of a particular type. However, if you prefer, you can code these brackets after the variable name as shown in the third example.

The fourth and fifth examples create arrays of objects. More specifically, the fourth example creates an array of three String objects, and the fifth example creates an array of five Product objects.

If you know the size of an array at compile time, you can code the length of the array as a literal value as shown in the first five examples. Or, you can code the length as a constant as shown in the sixth example. However, if you don't know the size of the array until runtime, you can use a variable to specify its size as shown in the seventh example.

## The syntax for declaring and instantiating an array

### With two statements

```
type[] arrayName;
arrayName = new type[length];
```

### With one statement

```
type[] arrayName = new type[length];
```

### With one statement and brackets after the variable name

```
type arrayName[] = new type[length];
```

## An example that declares and instantiates an array of double values

### With two statements

```
double[] prices;
prices = new double[4];
```

### With one statement

```
double[] prices = new double[4];
```

### With one statement and brackets after the variable name

```
double prices[] = new double[4];
```

## Other examples

### An array of String objects

```
String[] titles = new String[3];
```

### An array of Product objects

```
Product[] products = new Product[5];
```

### Code that uses a constant to specify the array length

```
final int TITLE_COUNT = 100;
String[] titles = new String[TITLE_COUNT];
```

### Code that uses a variable to specify the array length

```
System.out.print("Enter number of titles: ");
int titleCount = Integer.parseInt(sc.nextLine());
String[] titles = new String[titleCount];
```

## Description

- An *array* can store multiple primitive types such as int values or multiple reference types such as String objects. An *element* is one of the items in an array.
- To create an array, you must declare a variable of the correct type and instantiate an array object that the variable refers to. You can declare and instantiate the array in separate statements, or you can combine the declaration and instantiation into a single statement.
- To declare an array variable, you code a set of empty brackets after the type or the variable name.
- To instantiate an array, you use the new keyword and specify the *length*, or *size*, of the array in brackets following the array type.
- When you instantiate an array of primitive types, numeric types are set to zeros and boolean types to false. When you create an array of reference types, they are set to nulls.

Figure 10-1 How to create an array

## How to assign values to the elements of an array

Figure 10-2 shows how to assign values to the elements of an array. As the syntax at the top of this figure shows, you refer to an element in an array by coding the array name followed by an *index* in brackets. The index must be an int value starting at 0 and ending at one less than the size of the array. In other words, 0 refers to the first element in the array, 1 refers to the second element, 2 refers to the third element, and so on.

The first example in this figure shows how to assign values to the elements in an array by coding one statement per element. The first example creates an array of 4 double values. Then, it assigns a literal value to each element. In this example, the first element holds the value 14.95, the second holds 12.95, the third holds 11.95, and the fourth holds 9.95.

If you specify an index that's outside of the range of the array, Java throws an ArrayIndexOutOfBoundsException. For instance, the commented out line at the end of the first example refers to the element with index number 4. Since this array only has four elements, this statement throws this exception. Although you can catch this exception, it's better to write your code so it avoids using indexes that are out of bounds. To do that, you can check the length of the array as shown in the next figure.

The second and third examples work similarly to the first example. However, the second example creates an array that holds 3 String objects and initializes those strings. Similarly, the third example creates an array that holds 2 Product objects and initializes those objects. To do that, this code uses the getProduct method of the ProductDB class to return a Product object that corresponds with the specified product code.

After the third example, this figure shows how to create an array and assign values to the elements of the array in one statement. Here, you declare the array variable as usual. Then, you use the special assignment syntax to assign the initial values. With this syntax, you simply list the values you want assigned to the array within braces following the equals sign. Then, the number of values you list within the braces determines the size of the array that's created. The last three examples show how to use this special syntax to create the same arrays that were created by the first three examples in this figure.

### The syntax for referring to an element of an array

```
arrayName[index]
```

### Examples that assign values by accessing each element

#### Code that assigns values to an array of double types

```
double[] prices = new double[4];
prices[0] = 14.95;
prices[1] = 12.95;
prices[2] = 11.95;
prices[3] = 9.95;
//prices[4] = 8.95;     // this would throw ArrayIndexOutOfBoundsException
```

#### Code that assigns values to an array of String types

```
String[] names = new String[3];
names[0] = "Ted Lewis";
names[1] = "Sue Jones";
names[2] = "Ray Thomas";
```

#### Code that assigns objects to an array of Product objects

```
Product[] products = new Product[2];
products[0] = ProductDB.getProduct("java");
products[1] = ProductDB.getProduct("jsp");
```

### The syntax for creating an array and assigning values in one statement

```
type[] arrayName = {value1, value2, value3, ...};
```

### Examples that create an array and assign values in one statement

```
double[] prices = {14.95, 12.95, 11.95, 9.95};
String[] names = {"Ted Lewis", "Sue Jones", "Ray Thomas"};
Product[] products = {
        ProductDB.getProduct("java"),
        ProductDB.getProduct("jsp")
};
```

### Description

- To refer to the elements in an array, you use an *index* that ranges from zero (the first element in the array) to one less than the number of elements in the array.
- If you specify an index that's less than zero or greater than the upper bound of the array, Java throws an ArrayIndexOutOfBoundsException.
- You can instantiate an array and provide initial values in a single statement by listing the values in braces. The number of values you provide determines the size of the array.

Figure 10-2 How to assign values to the elements of an array