Chapter 7 - Arrays

Last updated: Tuesday, December 3, 2024

IntelliJ Chapter 07 Source Code

1. Open `Source Code\Chapter 07` as a project in IntelliJ.
2. Open the `CalcAverage` file.
3. Define a Project JDK.
4. Delete **InvalidSubscript** file (or, comment out the For Loop).
5. Run the `CalcAverage` file.

Intro to Arrays

- Primitive variables are designed to hold only one value at a time.
- Arrays allow us to create a collection of like values that are indexed.
- An array can store any type of data but only one type of data at a time.
- An array is a list of data elements.

An array is an object so it needs an object reference:

```
// Declare a reference to an array that will hold integers.
int[] numbers;
```

The next step creates the array and assigns its address to the numbers variable:

```
// Create a new array that will hold 6 integers.
numbers = new int[6];
```

Array element values are initialized to 0.
Array indexes always start at 0.

You can declare an array reference and create it in the same statement.

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

Arrays may be of any type (including objects):

```java
char[] letters = new char[26];
double[] temperatures = new double[7];
String[] daysOfWeek = new String[7];
```

The array size must be a non-negative number. It may be a literal value, a constant, or variable.

```java
final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```

Once created, an array size is fixed and cannot be changed. You cannot add or remove items from an array after it is initialized. You can, however, do that with a different object: **ArrayList**. (For more information, see Slides 69-79.)

Initialize Arrays with Custom Values

In the above examples, when we initialize an array using the following syntax:

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

the array is created using the default values of the data type. This works the same as in a Object class's Default Constructor, where fields are initialized to the default value of their data type. So the above six-element array has the following values: `{0, 0, 0, 0, 0, 0}`.

You can initialize an array with custom values by specifying them in curly brackets, like so:

```java
int[] daysInMonth = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};

String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};

char[] possibleGrades = {'F', 'D', 'C', 'B', 'A'};
```

Array Length

Arrays are objects which provide a public **field** named `length`. The length of an array can be obtained via its `length` constant field (which is defined as `public final int length`):

```java
int numberOfMonths = daysInMonth.length;
System.out.println(numberOfMonths)
// 12
```

```
System.out.println(daysOfWeek.length);
// 7
```

Note: Array `length` is a **field**, not a **method**. String has a **method** called `length()`. In an Array object, you access the public field. In a String, you access the public method, which uses parentheses.

```
char[] possibleGrades = {'F', 'D', 'C', 'B', 'A'};
// Access the Array object's public field: length
System.out.println(possibleGrades.length);

String s = "Sarah";
// Access the String object's public method: length()
System.out.println(s.length());
```

Accessing Elements of an Array

In our example array:

```
String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};
```

Array elements are accessed by:

- the array name: `daysOfWeek`
- An index position within square brackets, for example: `[0]`

```
System.out.println(daysOfWeek[0])
// Sunday
System.out.println(daysOfWeek[6])
// Saturday
```

Array elements can be treated as any other variable. You can also assign or re-assign the values in the array using the following syntax:

```
String[] cities = new String[3];
cities[0] = "Berlin"
cities[1] = "Beijing"
cities[2] = "Boston"

System.out.println(cities[0])
// Berlin
```

```
System.out.println(cities[1])
// Beijing
System.out.println(cities[2])
// Boston

cities[0] = "Bogota"
cities[1] = "Beirut"
cities[2] = "Bengaluru"

System.out.println(cities[0])
// Bogota
System.out.println(cities[1])
// Beirut
System.out.println(cities[2])
// Bengaluru
```

Alternatively, I can create an array with initialized values, and access and re-assign elements using same syntax:

```
String[] cities = { "Berlin", "Beijing", "Boston"};
System.out.println(cities[0])
// Berlin
cities[0] = "Bogota"
System.out.println(cities[0])
// Bogota
```

Important! Bounds Checking

Array indexes always start at `[0]` and continue to `[array.length - 1]`.

The `cities` array above was initialized with length `3`, but the final element in the array is `cities[2]`.

Because **Array indexes always start at 0**, the last element in the array is **always** located at the `[length-1]` position.

If you try to access an array element that is at `array.length` position or greater, you will trigger an `ArrayIndexOutOfBoundsException`. Here's an example in code:

```
String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};
System.out.println(daysOfWeek.length);
// 7
System.out.println(daysOfWeek[0]);
```

```
// Sunday
System.out.println(daysOfWeek[6]);
// Saturday
System.out.println(daysOfWeek[daysOfWeek.length - 1]);
// Saturday
System.out.println(daysOfWeek[7]);
//Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7
out of bounds for length 7
//      at ArrayPractice.main(ArrayPractice.java:11)
```

Also notice that array index positions can be accessed using variables. You can always access the last element in an array using the `[array.length - 1]` subscript.

Looping through Array Elements

Because Arrays have a known, fixed number of elements, looping through array elements is an ideal use case for a **for loop**. Here's the generic formula:

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

In for loops, it is typical to use i, j, and k as counting variables. It might help to think of **i** as representing the word **index**.

Here's an example and explanation:

```
String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};

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

Here's what's happening:

1. Initialize a counter (`i`) to `0`.
2. Evaluate test condition: if `i` is less-than the length of the array, execute the statement, which prints the array element at position `i`.
    1. `0 < 7`, prints `daysOfWeek[0]` which is `Sunday`. Increment `i`.
    2. `1 < 7`, prints `daysOfWeek[1]` which is `Monday`. Increment `i`.
    3. `2 < 7`, prints `daysOfWeek[2]` which is `Tuesday`. Increment `i`.

4. etc.. etc..
5. `7 < 7` is not true. Loop exits.

You can also use for loops to assign array values:

```
int[] squares = new int[20];
for (int i=0; i < squares.length; i++) {
        squares[i] = i * i;
}
System.out.println(squares[9]);
// 81
```

or use for loops to re-assign values:

```
String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};
System.out.println(daysOfWeek[6]);
// Saturday

for (int i=0; i < daysOfWeek.length; i++) {
        daysOfWeek[i] = daysOfWeek[i].replace("day", "yay").toUpperCase();
}
System.out.println(daysOfWeek[6]);
// SATURYAY
```

It is very easy to be off-by-one when accessing arrays. When looping through arrays, use **less than** and **array.length** to be safe.

Tip: Avoid Printing "Item #0" to User

Computers start counting at 0, but humans start at 1.

- The problem for your user is that it's odd to see "Enter ingredient #0: "
- The problem for the computer is that if you initialize your Array For Loop at 1, you won't add a value to the first item in the array (which is position 0).

The solution is to use 0 as the initial value of a for loop, and when you print messages to the user, use `(i + 1)`.

```
for (int i=0; i < ingredientsList.length; i++){
        System.out.print("Enter ingredient #" + (i+1) + ": ");
        ingredientsList[i] = kbd.next();
}
```

```
// Prints "Enter ingredient #1"
// Input is written to ingredientsList[0]
// and so forth..
```

This way, your user sees a number that makes sense to them, yet you're still writing to the correct position in the array.

Simplified Array for loop (read-only)

There's a simplified array processing syntax for reading elements only. You create a counter variable of the same data type as the array to hold each element (one at a time) while the loop executes the statement and then accesses the next element.

You can't re-assign elements this way because you don't refer to elements by their index, but rather by their value directly. The syntax is like this:

```
int[] numbers = {3, 6, 9};
for (int number : numbers) {
  System.out.println("The current number is " + number);
}
// The current number is 3
// The current number is 6
// The current number is 9

String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday"};
for (String day : daysOfWeek) {
        System.out.println(day);
}
```

User-specified Array length and element values

While Arrays are fixed in length, you can allow a user to specify the size and elements of an array using a Scanner object.

The following code prompts a user to enter the number of students, then allows the user to enter a name for each student:

```
int numberOfStudents;
String[] students;

Scanner keyboard = new Scanner(System.in);
System.out.print("How many students do you have? ");
numberOfStudents = keyboard.nextInt();
```

```
students = new String[numberOfStudents];

for (int i=0; i < students.length; i++){
        System.out.print("Name of student #" + i + ": ");
        students[i] = keyboard.nextLine();
}
```

Passing Arrays as Arguments to Methods

Arrays are objects. Their references can be passed to methods like any other object reference variable.

```
public static void main(String[] args) {
        String[] daysOfWeek = {"Sunday", "Monday", "Tuesday", "Wednesday",
"Thursday", "Friday", "Saturday"};
        printArray(daysOfWeek)
}

public static void printArray(String[] array){
        for (int i=0; i < array.length; i++){
                System.out.println(array[i]);
        }
}
```

Returning Arrays from Methods

Arrays can be a method return type modifier.

For example, if you had a method where you asked a user to enter ingredients, you can return an array of Strings by setting the return type as `String[]`:

```
public static String[] getIngredientsFromUser(){
        Scanner kbd = new Scanner(System.in);
        System.out.print("How many ingredients are you using? ");
        int numIngredients = kbd.nextInt();

        String[] ingredientsList = new String[numIngredients];
        for (int i=0; i < ingredientsList.length; i++){
                System.out.print("Enter ingredient #" + (i+1) + ": ");
                ingredientsList[i] = kbd.next();
        }
        return ingredientsList;
}
```

This method returns the array of ingredients that you can assign to a String array variable (the following example also loops through the array using the simplified array for loop syntax):

```
public static void main(String[] args) {
        String[] ingredients = getIngredientsFromUser();
        System.out.print("The ingredients in your sandwich are ");
        for (String ingredient : ingredients){
                System.out.print(ingredient + " ");
        }
}
```

Stray observation: main(String[] args)

You have survived three months of Java instruction, you are now ready to learn the secret meaning of `public static void main(String[] args)`.

You can specify an array of strings that you can refer to as "args" and a positional index in any executable Java program. For example, you have a String array available to you by default. Here's how you access elements 1 and 2 from that array:

```
public class StringArguments {
        public static void main(String[] args) {
                System.out.println(args[0]);
                System.out.println(args[1]);
        }
}
```

However, you can only pass in these arguments when running Java on the command line, and you must provide them yourself or you'll get an `ArrayIndexOutOfBoundsException`. The way you pass these arguments using the command line is like this (and the output):

```
java StringArguments.java "hi" "world"
hi
world
```

Alternatively, you can compile the program and then run it:

```
javac .\StringArguments.java
java StringArguments.java "hi" "world"
```

```
hi
world
```

Array of arrays: 2D arrays

[not on final exam]
(Slides 47 - 60)
An array of arrays in Java, also known as a multidimensional array, is a way to store data in a structured format. This is used in mathematical matrices, representing grids, or working with tabular data, like an Excel sheet.

Declare

You can declare an array of arrays using the following syntax:

```java
int[][] arrayName;
```

This declares `arrayName` as an array of arrays of integers.

Initialize

You can initialize an array of arrays using the `new` keyword:

```java
int[][] arrayName = new int[3][4];
```

This creates a 2D array with 3 rows and 4 columns.
Alternatively, you can initialize it with specific values:

```java
int[][] arrayName = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
};
```

Access elements

You can access elements in a 2D array using row and column indices:

```java
int value = arrayName[1][2];
// Accesses the element in the second row, third column
```

Iterating over elements

You can use nested loops to iterate over the elements of a 2D array:

```java
for (int i = 0; i < arrayName.length; i++) {
    for (int j = 0; j < arrayName[i].length; j++) {
        System.out.print(arrayName[i][j] + " ");
    }
    System.out.println();
}
```

Example: `lengths.java`

Ragged Arrays

Java allows you to create ragged arrays, where each row can have a different number of columns:

```java
int[][] raggedArray = new int[3][];
raggedArray[0] = new int[2];
raggedArray[1] = new int[3];
raggedArray[2] = new int[1];
```