Chapter 7. Arrays

Arrays are a convenient and necessary part of any computing language, as they facilitate the grouping of data items together as an single indexed unit. In this chapter, you get an opportunity to exercise and enhance your skills with arrays in Java. Among the exercises presented in this chapter are exercises on array declaration, creation, initialization, assignment, access, copying, reversal, subset manipulation, sorting and element searches. Exercises on multi-dimensional arrays are also presented.

In addition to the built-in syntax for array handling, recourse is made heavily to the methods of the class `Arrays`.

1.	In Java, what is an *array*?	An array is a data element that has an indexed list of items (primitives or objects) that are of the *same type. The array index is an integer index. *If you want to put objects that are of different types into an array, you have to cast the objects to the type of the array (if possible in the given case). In that sense the items in the array are of the same type.
2.	Are arrays descended from the class `Object`?	Yes they are, as arrays are objects and the ultimate ancestor class of objects in Java is the class `Object`.
3.	I have an array of integers defined as follows: `int[] intArray = new int[3];` Explain in words what each part of this declaration statement means.	
	1. **`int[]`** is the type of what we are going to define, which in this case is a reference variable to an array of integers. In a variable declaration, **`[]`** after the name of a type indicates that it is an array reference variable that will contain the specified type (`int` in this case). 2. **`intArray`** is the variable name we've chosen to give to this array object reference that contains elements of type `int`. 3. **`new int[3]`** means that we are asking the system to allocate space for an array of 3 integers. 4. **`=`** This is saying that the newly allocated array of integers is going to be known by the variable name **`intArray`** which appears on the left-hand side of the equals sign.	
4.	What is the 1st index of an array?	**0**. Arrays in Java always start at index 0.
5.	Write a line of code which declares an `int` array variable named `intArray`.	`int[] intArray;`
6.	Write a line of code which will instantiate an `int` array named `intArray` with a capacity for 20 integers.	`int[] intArray = new int[20];`
7.	Assign the value 2 to the 1st index of the array defined above.	`intArray[0] = 2; // Remember that the 1st index is 0`
8.	Assign the value 800 to the 1st index of the array above.	`intArray[0] = 800;` This simply overwrites what was previously set therein.
9.	Assign the value 17 to the **3rd** index of the array `intArray` defined earlier.	`intArray[2] = 17;`

10.	Write a program which instantiates an array of size 5 of each of the following types: `Float`, `float`, `Integer`, `int`, `double`, `boolean`.	`import java.util.*;`
		`public class PracticeYourJava {`
		`  public static void main(String[] args) {`
		`  Float[] fArray01 = new Float[5];`
		`  float[] fArray02 = new float[5];`
		`  Integer[] iArray01 = new Integer[5];`
		`  int[] iArray02 = new int[5];`
		`  double[] dArray01 = new double[5];`

<table>
<tr><td></td><td></td><td>

```
    boolean[] bArray01 = new boolean[5];
  }
}
```

</td></tr>
<tr><td>11.</td><td>Write a code fragment which will instantiate a <code>String</code> array that has a capacity for 10 <code>String</code> objects. Initialize the first 5 elements of the array with each of the following words respectively: <code>“Mary”, “had”, “a”, “little”, “lamb”</code>.</td><td>

```
String[] sArray01 = new String[10];
sArray01[0]= "Mary";
sArray01[1]="had";
sArray01[2]="a";
sArray01[3]="little";
sArray01[4]="lamb";
```

</td></tr>
<tr><td rowspan="2">12.</td><td colspan="2">Write a program which will print out the <code>length</code> of the array from the preceding exercise.
Hint: the <code>Length</code> field of arrays.</td></tr>
<tr><td colspan="2">

```
public class PracticeYourJava {
  public static void main(String[] args) {

    String[] sArray01 = new String[10];
    sArray01[0] = "Mary";   sArray01[1] = "had"; sArray01[2] = "a";
    sArray01[3] = "little"; sArray01[4] = "lamb";
    System.out.printf("Array length = %d\n", sArray01.length);
  }
}
```

Note: As seen from the answer above, the `length` of the array is the number of allocated spaces in the array, not the number of occupied spaces.
</td></tr>
<tr><td rowspan="2">13.</td><td colspan="2">Write a program which instantiates an integer array named <code>intArray01</code> that can hold 50 integers. Print out the length of the array using the <code>Length</code> field of arrays.</td></tr>
<tr><td colspan="2">

```
public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[50];
    int arrayLength = intArray01.length; // The length of the array
    System.out.printf("The length of the array is %d.\n", arrayLength);
  }
}
```

</td></tr>
<tr><td rowspan="2">14.</td><td colspan="2">State the difference between the following two initialization methods for an array:

```
int[] intArray01 = new int[5];
intArray01[0] = 10;
intArray01[1] = 11;
intArray01[2] = 12;
intArray01[3] = 13;
intArray01[4] = 14;
```

versus

```
int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };
```

</td></tr>
<tr><td colspan="2">In this example, there is no difference at all. The second method is quite convenient when you are hardcoding arrays into a program and also want the length of the array to match the number of elements that you are initializing it with. In this case (the second case), the compiler infers the desired number of elements of the array by counting the number of comma separated items in the braces.
The benefit of the first initialization method however is that we can declare the array to be of a size greater than the number of elements that we plan to initialize it with, as opposed to the second method where the compiler only allocates as many spaces to the array as the number of elements specified at instantiation.
Nonetheless, irrespective of whichever method is chosen, Java provides an indirect way to modify the number of allocated spaces in an already existing array. We will see this in a later exercise.</td></tr>
<tr><td rowspan="2">15.</td><td colspan="2">*Yes* or *No*: Does Java initialize the values in an array?</td></tr>
<tr><td colspan="2">Yes.
The reader should contrast this with the fact that Java does not initialize individual variables.</td></tr>
<tr><td rowspan="2">16.</td><td colspan="2">What are the elements of an array of numerical primitives initialized to by default?</td></tr>
<tr><td colspan="2">0.</td></tr>
</table>

17. What are the elements of an array of objects initialized to by default?

`null`

18. Explain what is wrong with the following code:

```
public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };
        System.out.printf("%d\n", intArray02[5]);
  }
}
```

The code attempts to print out the 6th element of an array that only has 5 elements defined. On running the code, the following exception will be produced: `java.lang.ArrayIndexOutOfBoundsException.`
Conclusion: Never attempt to access a non-existent array position.

19. Write a program which instantiates an `int[]` with 0 elements. Print the length of the array out.

```
public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[0];
    System.out.println("Length of intArray01 = " + intArray01.length);
  }
}
```

20. Explain the difference between the following two array declarations:

```
int[] intArray01;
int[] intArray02 = new int[0];
```

The first declaration is merely creating a variable that will hold an array; no array has been defined at all. The second declaration actually defines an empty array.

21. *Copying array contents*
Copy the following array to a new `int[]` named `intArray02`. Print the contents of the new array out.

```
int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };
```

Hint: `Arrays.copyOf`

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };
    int[] intArray02 = Arrays.copyOf(intArray01, intArray01.length);
    System.out.printf("%d, %d, %d, %d, %d \n", intArray02[0], intArray02[1], intArray02[2],
                                                  intArray02[3], intArray02[4]);
  }
}
```

22. Modify the solution to the preceding exercise to also check whether the contents of `intArray01` and `intArray02` are the same. Print out your findings.
Hint: `Arrays.equals`

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] { 10, 11, 12, 13, 14 };
    int[] intArray02 = Arrays.copyOf(intArray01, intArray01.length);

    boolean areEqual = Arrays.equals(intArray01, intArray02);
    if(areEqual == true)
      System.out.println("The arrays have the same content!");
    else
      System.out.println("The arrays do NOT have the same content!");
  }
}
```

23. *Copying part of an array to another*

We have an integer array `intArray01` that has 5 elements as follows:

```
[0] -    55
[1] -   747
[2] - 15000
[3] -    89
[4] -  2333
```

Copy the 1st three elements of `intArray01` into an array `intArray02`. Print the length of `intArray02` and its contents out.

Hint: `Arrays.copyOfRange`

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {500,747,15000,89,2333};
    int[] intArray02;

    // Now the 1st 3 elements of intArray01 to intArray02
    // Therefore we are index 0 to index 2

    intArray02 = Arrays.copyOfRange(intArray01, 0, 3);
    // Observe the assignment to intArray02. An array of the
    // exact length returned by the copy is returned.

    System.out.printf("intArray02's length is %d\n",
                                  intArray02.length);
    System.out.printf("intArray02's contents are %d, %d, %d \n",
                                  intArray02[0], intArray02[1],
                                  intArray02[2]);
  }
}
```

Notes: Observe that the length of the array returned is exactly the number of elements put into it.

24. We have an integer array `intArray01` which has 5 elements with the following content:

```
[0] -    55
[1] -   747
[2] - 15000
[3] -    89
[4] -  2333
```

Copy the *2nd to the 4th* elements of `intArray01` into the 3rd to 5th position of `intArray02` which also is an `int` array of 5 elements. Print the contents of `intArray02` out before and after the copy operation.

Hint: method `System.arraycopy`

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {55,747,15000,89,2333};
    int[] intArray02 = new int[intArray01.length];   //the
          //target array has to be >= the source in length

    System.arraycopy(intArray01, 0, intArray02, 2, 3);
    // The 2nd element in intArray01 is position [1],
    // the 3rd element in intArray02 is position [2].
    // and we are copying 3 elements.

    System.out.printf("intArray02's contents are now
                      %d,%d,%d,%d,%d\n", intArray02[0],
                      intArray02[1], intArray02[2],
                       intArray02[3], intArray02[4]);
  }
}
```

25. Rewrite the preceding program, using the `Arrays.toString` method to output the contents of the array.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {
    int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
    int[] intArray02 = new int[intArray01.length];
    //the target array has to be >= the source in length

    System.arraycopy(intArray01, 0, intArray02, 2, 3);
    // The 2nd element in intArray01 is position [1],
    // the 3rd element in intArray02 is position [2].
    // and we are copying 3 elements.

    System.out.printf("intArray02's contents are now %s\n", Arrays.toString(intArray02));
  }
}
```

26. Run the following program and state your observations on it and its output.

```
import java.util.*;
```

```
public class PracticeYourJava {
  public static void main(String[] args) {
    int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
    int[] intArray02;

    System.out.printf("intArray01's contents are currently %s\n", Arrays.toString(intArray01));

    intArray02 = Arrays.copyOfRange(intArray01, 0, 10);

    System.out.println("Length of intArray02 = " + intArray02.length);
    System.out.printf("intArray02's contents are %s\n", Arrays.toString(intArray02));
  }
}
```

Observations: In the call to the method `Arrays.copyOfRange`, the program states that it is going to copy the contents of `intArray01` to `intArray02`. However, the specified number of elements to "copy" is **10**, which is greater than the number of elements in the source array `intArray01`. Nevertheless, the code runs and we see that `intArray02` has a length of 10 elements! On looking at the contents of `intArray02` we observe that indeed the values in `intArray01` were copied to `intArray02` and the extra positions were filled with the default value of `0` for the `int[]` type.
This mechanism of specifying a greater length than the length of the source array is actually a way to use the method **`Arrays.copyOfRange`** to programmatically create an array of longer length than the source array. This same mechanism can be used to lengthen the source array itself by specifying the same array as source and destination (what really happens is that another array object would be created in the background with the same data as the original array and that the memory address of the new array object would be assigned to the variable name of the original array).

27. Shorten the following array to only its 1st three elements:

```
int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
```

Print out the length of `intArray01` to prove that its length has indeed been modified.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {55,747,15000,89,2333};
    intArray01 = Arrays.copyOfRange(intArray01, 0, 3); //source & destination arrays are the same
    System.out.println("New length of intArray01 = " + intArray01.length);
  }
}
```

28. *Extending the length/Increasing the capacity of an array*
Extend the length of the following array to ten elements:

```
int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
```

Print out the length of `intArray01` as well as its contents.
Hint: Read the solution to exercise 26.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {55,747,15000,89,2333};
    System.out.println("Current length of intArray01 = " + intArray01.length);

    intArray01 = Arrays.copyOfRange(intArray01, 0, 10);
    // Note that the source & destination arrays are the same

    System.out.println("New length of intArray01 = " + intArray01.length);
    System.out.printf("intArray01's contents are now  %s\n", Arrays.toString(intArray01));
    // Note that the extra elements are automatically set to the default value of the type
    // of the array.
  }
}
```

29. We have an integer array named `intArray01` initialized with the following 5 elements: `{17,42,43,8,23}`.
Write a program which extends the length of this array and then puts two more elements, namely the values 57 and 84 at the end of the array. Print out the length of the new array as well as its contents.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

   int[] intArray01 = new int[] {17,42,43,8,23};

   // We use an indirect mechanism to extend the length of the array;
   intArray01 = Arrays.copyOf(intArray01, intArray01.length + 2);
   // So we gave it the contents of intArray01 and told it to return the same data
   // in an array of length of intArray + 2.

   // Now put our desired data into the two new elements.
   intArray01[5] = 57;
   intArray01[6] = 84;

   System.out.printf("intArray01's new length is %d\n",  intArray01.length);
   System.out.printf("intArray01's contents are now  %s\n", Arrays.toString(intArray01));
  }
}
```

30. *Shortening the length/Decreasing the capacity of an array*
We have the following array: `int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};`
Programmatically modify this array to the following: `int[] intArray01 = new int[] {15000, 89, 2333};`
Print out the new length of the array as well as its contents.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {55,747,15000,89,2333};
    int startingIndex = 2;
    int finalIndex = intArray01.length;

    intArray01 = Arrays.copyOfRange(intArray01, startingIndex, finalIndex);
    System.out.println("New length of intArray01 = " + intArray01.length);
    System.out.printf("intArray01's contents are now  %s\n", Arrays.toString(intArray01));
  }
}
```

31. We've seen that when an array is created but not initialized, it is filled with the default value for its type. For example, an `int[]` is filled with the value `0`. However, there are times when `0` is a valid data point for us and so we wouldn't know whether we ourselves set particular entries to `0`, or whether it is the default value that is the content of any given array element. This being the case, what is the easiest solution with which to set the elements in an array to a value that is not a valid data point for our data so that we can tell whether or not we have set the value? For example, we might want to set the entries to `-1` as the default.
Hint: `Arrays.fill`

The solution is to fill the array with the desired flag value, using the `Arrays.fill` method.

32. Create an `int` array of length 5 and initialize each of its elements with the value -1. Print the array out afterwards.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[5];

    Arrays.fill(intArray01, -1);
    System.out.printf("intArray01's contents are now  %s\n", Arrays.toString(intArray01));
   }
}
```

33. Sort the following array and print its contents out: `int[] intArray01 = new int[] {55,747,15000,89,2333};`
Hint: `Arrays.sort`

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {
```

```
    int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
    Arrays.sort(intArray01);
    System.out.printf("intArray01's contents are now  %s\n", Arrays.toString(intArray01));
  }
}
```

34. Repeat the same exercise as above, this time using the `Arrays.parallelSort` method.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {55, 747, 15000, 89, 2333};
    Arrays.parallelSort(intArray01);
    System.out.printf("intArray01's contents are now  %s\n", Arrays.toString(intArray01));
  }
}
```

35. Briefly explain the difference between `Arrays.parallelSort` and `Arrays.sort`.

The difference between the two methods is that ***`Arrays.parallelSort`*** is designed to be able to sort using multiple threads. The result is that on multi-core or multi-threaded processors `Arrays.parallelSort` should be faster than `Arrays.sort`.

36. Determine using the `binarySearch` method of class `Arrays`, which array element contains the value `747` in the sorted array of the preceding exercise. (`binarySearch` only works on sorted arrays).

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[] intArray01 = new int[] {55,747,15000,89,2333};
    int valueToSearchFor = 747;
    int position;
    Arrays.sort(intArray01);
    position = Arrays.binarySearch(intArray01, valueToSearchFor);
    if(position >= 0)
      System.out.printf("The value %d was found at position %d\n", valueToSearchFor, position);
  }
}
```

37. I have the following `String` object: `s01 = "Hello how are you?"`. Convert this to an array of bytes. State how many elements are in the byte array.
Hint: ***`String.getBytes`***

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    String s01 = "Hello how are you?";
    byte[] strAsByteArray = s01.getBytes();
    System.out.printf("The length of the byte array is %d\n", strAsByteArray.length);
  }
}
```

Important note: The conversion is done in the default charset of the system you are running your program on. We do not cover the concept of charsets in this book, however you should be aware of the concept of charsets, as the selected/specified charset can affect the equivalent number of bytes that a given string converts to.

38.	*Matrices/Multi-dimensional arrays* Write a line of code to declare a variable `matrixA` that can refer to a 2-dimensional `int` array/matrix.	`int[][] matrixA;` The number of braces (two in this case) is what tells the compiler how many dimensions the multi-dimensional array in question will have.
39.	Write a line of code to instantiate a 2-dimensional integer matrix named `matrixA` of dimensions 2 x 3.	`int[][] matrixA = new int[2][3];`

(i.e. 2 rows and 3 columns).

40. Write the appropriate line of code to declare a 5-dimensional integer array/matrix variable named `matrixB`.

```
int[][][][][] matrixB;
```

41. Write a line of code that shows the instantiation of a 5-dimensional integer matrix named `matrixB` of dimensions 10 x 4 x 3 x 5 x 12.

```
int[][][][][] matrixB = new int[10][4][3][5][12];
```

42. Instantiate two *2 x 3* integer matrices, with variable names `matrix1` and `matrix2` respectively and initialize them as follows:

matrix1

1	2	3
4	5	6

matrix2

5	5	8
3	1	3

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[][] matrix1 = new int[2][3];
    matrix1[0][0] = 1;  matrix1[0][1] = 2; matrix1[0][2] = 3;
    matrix1[1][0] = 4;  matrix1[1][1] = 5; matrix1[1][2] = 6;

    int[][] matrix2 = new int[2][3];
    matrix2[0][0] = 5;  matrix2[0][1] = 5; matrix2[0][2] = 8;
    matrix2[1][0] = 3;  matrix2[1][1] = 1; matrix2[1][2] = 3;
  }
}
```

OR, in compact form,

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[][] matrix1 = new int[][] { { 1, 2, 3 }, { 4, 5, 6 } };
    int[][] matrix2 = new int[][] { { 5, 5, 8 }, { 3, 1, 3 } };
    // Observe that the data is entered row by row,
    // comma delimited in curly braces per row.
  }
}
```

43. Add the contents of the arrays of the preceding exercise, putting the result into a new matrix, `matrix3`.

```
import java.util.*;

public class PracticeYourJava {
  public static void main(String[] args) {

    int[][] matrix1 = new int[][] { { 1, 2, 3 }, { 4, 5, 6 } };
    int[][] matrix2 = new int[][] { { 5, 5, 8 }, { 3, 1, 3 } };

    int[][] matrix3 = new int[2][3];
    matrix3[0][0] = matrix1[0][0] + matrix2[0][0];
    matrix3[0][1] = matrix1[0][1] + matrix2[0][1];
    matrix3[0][2] = matrix1[0][2] + matrix2[0][2];
    matrix3[1][0] = matrix1[1][0] + matrix2[1][0];
    matrix3[1][1] = matrix1[1][1] + matrix2[1][1];
    matrix3[1][2] = matrix1[1][2] + matrix2[1][2];
  }
}
```

E1 Note the method `Arrays.asList(T... a)`. We will see it later in the chapter on Collections.

There are a number of other built-in different ways and forms in which arrays and more advanced variants thereof are presented in Java; these will be seen in the chapter entitled "Collections". Also, we will see further manipulations of arrays in the next chapter.