

Lesson 6

Arrays

Absolute Java

Walter Savitch

Array

- In Java, an array is a special kind of object.
- It is often more useful to think of it as a collection of variables all of the same type.
- The elements in the array are stored on consecutive memory locations.
- It is also known as a subscripted variable. E.g. $A[i]$ is the i th element of the array (i is subscript with variable A).

One Dimensional Array

- One-dimensional array has one subscript.
- Each element of an array occupies separate consecutive memory location, and is referred by its subscript.
- In Java, the subscript of an array starts from 0.
- If A is an array, then A[0] is the first element of an array, i.e. in general, the *i*th element is at position (i-1).

Syntax for declaring an Array:

Base_Type[] Array_Name = new Base_Type[Length];

- The *Length* may be given as any expression that evaluates to a nonnegative integer.
- *Length* can be an **int** variable.

One Dimensional Array

- One-dimensional array has one subscript.
- Each element of an array occupies separate consecutive memory location, and is referred by its subscript.
- In Java, the subscript of an array starts from 0.
- If A is an array, then A[0] is the first element of an array, i.e. in general, the *i*th element is at position (i-1).

Syntax for declaring an Array:

Base_Type[] Array_Name = new Base_Type[Length];

- The *Length* may be given as any expression that evaluates to a nonnegative integer.
- *Length* can be an **int** variable.

One Dimensional Array

- Examples:

```
double[ ] temperature = new double[7];
```

```
double[] score = new double[5];
```

```
char[] line = new char[80];
```

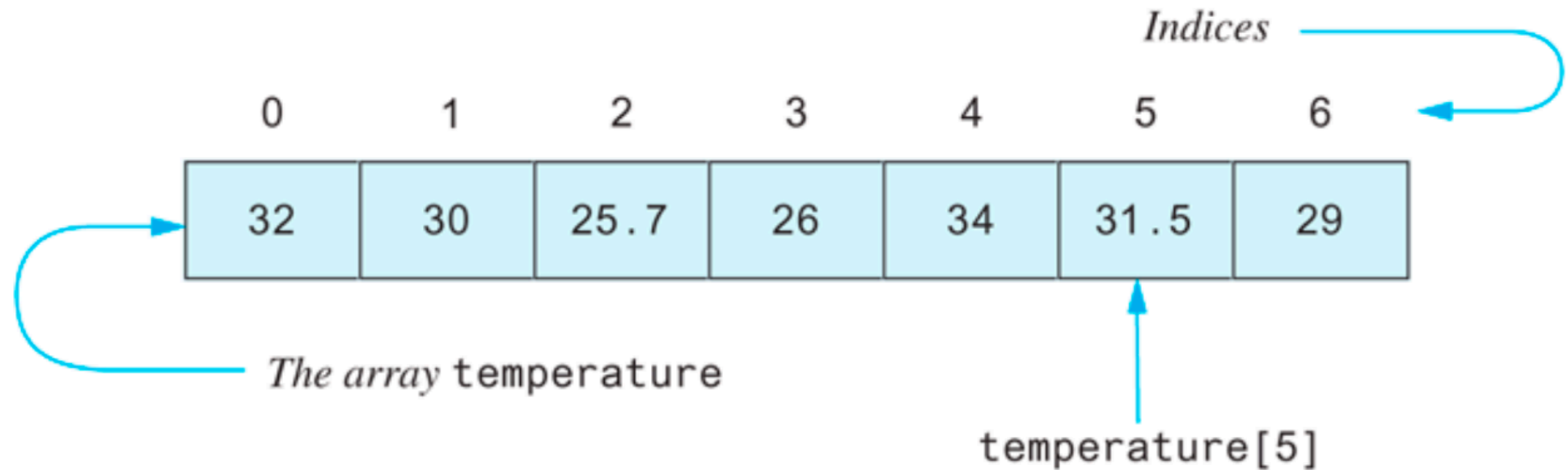
```
double[] reading = new double[300];
```

```
Person[] specimen = new Person[100]; //Person is a class
```

- **Index:** An index is an integer expression that indicates an array element.
- In Java, the indices of an array always start with 0.
- They never start with 1 or any number other than 0.

One Dimensional Array

A Common Way to Visualize an Array



One Dimensional Array

- We can use named constants as the size of the array.
- **For example:**

```
public static final int NUMBER_OF_READINGS = 100;  
int[ ] pressure = new int[NUMBER_OF_READINGS];
```

One Dimensional Array

- **Specifying array size at declaration time is not important:**
- Java allocates memory for an array, as well as for any other object, at execution time.
- So if you do not know how large to make an array when you write a program, you can read the array's length from the keyboard.
- **For Example:**

```
System.out.println("How many temperatures do you have?");
```

```
int size = keyboard.nextInt();
```

```
double[ ] temperature = new double[size];
```


One Dimensional Array

- **Using expressions as index:**
- You can also use any expression that evaluates to an appropriate integer when naming an indexed variable of an array.
- **For Example:**

```
int point = 2;
```

```
temperature[point + 3] = 32;
```

```
System.out.println("Temperature is " + temperature[point + 3]);
```

One Dimensional Array

- Each of the indexed variables of an array can be used just like any other variable .
- For example, all of the following are allowed in Java:

```
score[3] = 32;  
score[0] = score[3] + 10;  
System.out.println(score[0]);
```

One Dimensional Array

- **Instance Variable length:** An array is a kind of object has only one public instance variable, namely the variable **length**, which is equal to the length of the array.
- **For Example:**

```
Species[ ] entry = new Species[20];
```

```
System.out.println("Length of array=" + entry.length);
```

This statement will display:

```
Length of array=20
```

One Dimensional Array

- **Instance Variable length:** An array is a kind of object has only one public instance variable, namely the variable **length**, which is equal to the length of the array.
- **For Example:**

```
Species[ ] entry = new Species[20];
```

```
System.out.println("Length of array=" + entry.length);
```

This statement will display:

```
Length of array=20
```

One Dimensional Array

- **Length is a final variable:** We cannot assign a value to the instance variable length, as it is a final variable.
- For example, the following attempt to change the size of an array is invalid:

```
entry.length = 10; //Illegal!
```

One Dimensional Array

- **Array Indices Must Be Within Bounds to Be Valid:** Since the index of the first element in a Java array is always 0, the last index number is not the length of the array, but is one less than the length of the array.
- Be sure that your indices stay within this range.
- **An array index that is less than 0 or greater than or equal to the size of the array will cause an error message during program execution.**

One Dimensional Array

- **Initializing Arrays:** An array can be initialized at the time that it is declared.
- To do this, you enclose the values for the individual indexed variables in braces and place them after the assignment operator, as in the following example:

```
double[ ] reading = {3.3, 15.8, 9.7};
```

- The size of the array—that is, its length—is set to the minimum that will hold the given values.

One Dimensional Array

- **Arrays can be used as instance variables in classes:** Methods can have an indexed variable or an entire array as an argument and can return an array.
- In short, arrays can be used with classes and methods just as other objects can.

One Dimensional Array

- **Automatic initialization:** If you do not initialize the elements of an array, they will automatically be initialized to a default value for the base type.
- For numeric types, the default value is the zero of the type.
- For base type char, the default value is the non printable zeroth character (char)0, not the space character.
- For the type boolean, the default value is false.
- For class types, the default value is null.
- For example, if you do not initialize an array of doubles, each element of the array will be initialized to 0.0.

One Dimensional Array

Check your Understanding

1. In the array declaration

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

what is

- a. the array name?
- b. the base type?
- c. the length of the array?
- d. the range of values an index accessing this array can have?
- e. one of the indexed variables (or elements) of this array?

One Dimensional Array

Check your Understanding

1. In the array declaration

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

what is

- a. the array name?
- b. the base type?
- c. the length of the array?
- d. the range of values an index accessing this array can have?
- e. one of the indexed variables (or elements) of this array?

One Dimensional Array

Check your Understanding

2. In the array

```
double[] score = new double[10];
```

what is

- a. the value of score.length?
- b. the first index of score?
- c. the last index of score?

3. What is the output of the following code?

```
char[] letter = {'a', 'b', 'c'};  
for (int index = 0; index < letter.length; index++)  
System.out.print(letter[index] + ", ");
```

One Dimensional Array

- **An Array of Characters Is Not a String:** An array of characters, is conceptually a list of characters; therefore, it is conceptually like a string:

```
char[] a = {'A', ' ', 'B', 'i', 'g', ' ', 'H', 'i', '!'};
```

- However, an array of characters, such as `a`, is not an object of the class `String`. In particular, the following is illegal in Java:

```
String s = a;
```

- Similarly, you cannot normally use an array of characters, such as `a`, as an argument for a parameter of type `String`.

One Dimensional Array

- **Converting Array of character to String:** To convert an array of characters to an object of type String, the class String has a constructor that has a single parameter of type char[].
- So, you can obtain a String value corresponding to an array of characters, such as a, as follows:

String s = new String(a);

- The object s will have the same sequence of characters as the array a.
- The object s is an independent copy; any changes made to a will have no effect on s.
- Note that this always uses the entire array a.

One Dimensional Array

- There is also a `String` constructor that allows you to specify a subrange of an array of characters `a`.
- For example,
`String s2 = new String(a, 2, 3);`

produces a `String` object with 3 characters from the array `a` starting at index 2. So, if `a` is as above, then

```
System.out.println(s2);
```

outputs

Big

One Dimensional Array

- There is also a `String` constructor that allows you to specify a subrange of an array of characters `a`.
- For example,
`String s2 = new String(a, 2, 3);`

produces a `String` object with 3 characters from the array `a` starting at index 2. So, if `a` is as above, then

```
System.out.println(s2);
```

outputs

Big

One Dimensional Array

- Although an array of characters is not an object of the class String, it does have some things in common with String objects.
- For example, you can output an array of characters using println, as follows:

System.out.println(a);

which produces the output

A Big Hi!

Array are objects in Java

- **Arrays Are Objects:** In Java, arrays are considered to be objects.
- **Array Types Reference Types:** A variable of an array type holds the address of where the array object is stored in memory. This memory address is called a **reference** to the array object.
- For example:
`double[] a = new double[10];`
- Notice that this is almost exactly the same as the way that we view objects of a class type.
- Every array has an instance variable named **length**, which is a good example of viewing an array as an object.
- Because an array type is a reference type, the behaviors of arrays with respect to assignment =, ==, and parameter passing mechanisms are the same as for classes.

Array are objects in Java

- **Arrays with a Class Base Type:** The base type of an array can be of any type, including a class type.
- For example, suppose **Date** is a class and consider the following:
Date[] holidayList = new Date[20];
- This creates the 20 indexed variables (**holidayList[0]**, **holidayList[1]**, ..., **holidayList[19]**) of type **Date**. This does not create 20 *objects* of type Date. (The index variables are automatically initialized to **null**, not to an object of the class **Date**.)
- Like any other variable of type Date, the indexed variables require an invocation of a constructor using new to create an object.

- **Example:**

```
Date[] holidayList = new Date[20];  
for (int i = 0; i < holidayList.length; i++)  
    holidayList[i] = new Date( );
```

Array are objects in Java

- If you omit the **for** loop (and do not do something else more or less equivalent), then when you run your code, you will undoubtedly get an error message indicating a “**null pointer exception.**”
- If you do not use **new** to create an object, an indexed variable like **holidayList[i]** is just a variable that names no object and hence cannot be used as the calling object for any method.

Array are objects in Java

- **Array Parameters:** You can use both array indexed variables and entire arrays as arguments to methods, although they are different types of parameters.

- Example :

```
myMethod(a[3]);
```

Array are objects in Java

- **Entire Arrays as Arguments to a Method** : A parameter can represent an entire array.
- **Example :**

```
public class SampleClass
{
    public static void incrementArrayBy2(double[] anArray)
    {
        for (int i = 0; i <anArray.length; i++)
            anArray[i] = anArray[i] + 2;
    }
    <The rest of the class definition goes here>
}
```

Array are objects in Java

- No square brackets are used when you give an entire array as an argument to a method.
- An array type is a reference type just as a class type is, so, as with a class type argument, a method can change the data in an array argument.

Array are objects in Java

- **Methods That Return an Array:** In Java, a method may return an array.
- You specify the return type for a method that returns an array in the same way that you specify a type for an array parameter.
- **Example:**

```
public static char[] upperCaseVersion(char[] a)
{
    char[] temp = new char[a.length];
    int i;
    for (i = 0; i < a.length; i++)
        temp[i] = Character.toUpperCase(a[i]);
    return temp;
}
```


Arguments for the Method main

- The heading for the main method of a program has a parameter for an array of base type of String:

public static void main(String[] args)

- The identifier **args** is a parameter of type **String[]**.
- Because args is a parameter, it could be replaced by any other non keyword identifier.
- If no argument is given when you run your program, then a default empty array of strings is automatically provided as a default argument to main when you run your program.

Arguments for the Method main

- It is possible to run a Java program in a way that provides an argument to plug in for this array of String parameters.
- You provide any number of string arguments when you run the program, and those string arguments will automatically be made elements of the array argument that is plugged in for args (or whatever name you use for the parameter to main).
- **This is normally done by running the program from the command line of the operating system.**

Arguments for the Method main

- For example:

```
java YourProgram Do Be Do
```

- This will set args[0] to "Do", args[1] to "Be", args[2] to "Do", and args.length to 3.
- These three indexed variables can be used in the method main, as in the following sample program:

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

Partially Filled Arrays

- Often the exact size needed for an array is not known when a program is written, or the size may vary from one run of the program to another.
- One common and easy way to handle this situation is to declare the array to be of the largest size the program could possibly need.
- The program is then free to use as much or as little of the array as needed.

Multidimensional Arrays

- Arrays having multiple indices are handled in much the same way as one-dimensional arrays.
- The following statement declares the name table and creates a two-dimensional array :

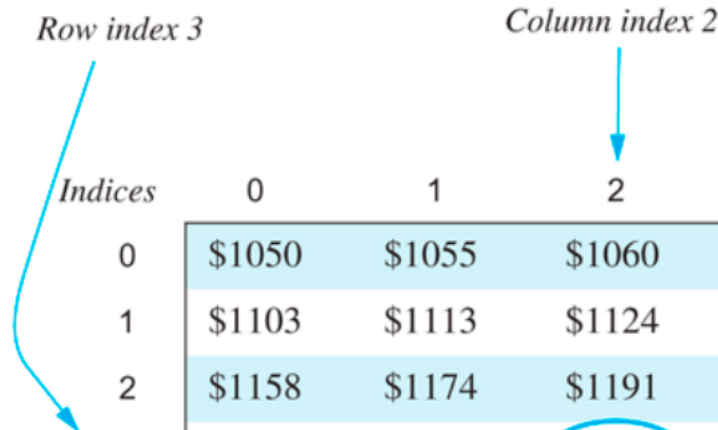
```
int[ ][ ] table = new int[10][6];
```

- This is equivalent to the following two statements:

```
int[ ][ ] table;  
table = new int[10][6];
```

Multidimensional Arrays

Row and Column Indices for an Array Named table



		Column index 2					
Row index 3							
Indices		0	1	2	3	4	5
0		\$1050	\$1055	\$1060	\$1065	\$1070	\$1075
1		\$1103	\$1113	\$1124	\$1134	\$1145	\$1156
2		\$1158	\$1174	\$1191	\$1208	\$1225	\$1242
3		\$1216	\$1239	\$1262	\$1286	\$1311	\$1335
4		\$1276	\$1307	\$1338	\$1370	\$1403	\$1436
5		\$1340	\$1379	\$1419	\$1459	\$1501	\$1543
6		\$1407	\$1455	\$1504	\$1554	\$1606	\$1659
7		\$1477	\$1535	\$1594	\$1655	\$1718	\$1783
8		\$1551	\$1619	\$1689	\$1763	\$1838	\$1917
9		\$1629	\$1708	\$1791	\$1877	\$1967	\$2061

`table[3][2]` has
a value of 1262

Multidimensional Arrays

- Arrays having more than one index are generally called multidimensional arrays.
- More specifically, an array that has n indices is said to be an n -dimensional array.
- Thus, an ordinary one-index array is a one-dimensional array.
- Although arrays having more than two dimensions are rare, they can be useful for some applications.

Multidimensional Arrays

- The syntax is almost identical to the syntax we use for the one-dimensional arrays.
- The only difference is that we added a second pair of square brackets in two places, and we gave a number specifying the size of the second dimension, that is, the number of columns.
- You can have arrays with any number of indices.
- To get more indices, you just use more square brackets in the declaration.
- Indexed variables for multidimensional arrays are just like indexed variables for one-dimensional arrays, except that they have multiple indices, each enclosed in a pair of square brackets.

Multidimensional Arrays

- The Java compiler represents a multidimensional array as several one-dimensional arrays.
- For example, consider the two-dimensional array
`int[][] table = new int[10][6];`
- The array table is in fact a one-dimensional array of length 10, and its base type is the type `int[]`.
- In other words, multidimensional arrays are arrays of arrays.
- When using length with a multidimensional array, you need to think in terms of arrays of arrays.

```
for (int row = 0; row < table.length; row++)  
    for (int column = 0; column < table[row].length; column++)  
        table[row][column] = getBalance(1000.00, row + 1, (5 + 0.5 * column));
```

Multidimensional Arrays

- **Multidimensional Array Parameters and Returned Values:**Methods may have multidimensional array parameters and may have a multidimensional array type as the type for the value returned.
- The situation is similar to that of the one-dimensional case, except that you use more square brackets when specifying the type name.
- **Example:**

```
public static void showMatrix(int[][] a)
{
    int row, column;
    for (row = 0; row < a.length; row++)
    {
        for (column = 0; column < a[row].length; column++)
            System.out.print(a[row][column] + " ");
        System.out.println();
    }
}
```

Multidimensional Arrays

- **Multidimensional Array Parameters and Returned Values:**Methods may have multidimensional array parameters and may have a multidimensional array type as the type for the value returned.
- The situation is similar to that of the one-dimensional case, except that you use more square brackets when specifying the type name.
- **Example:**

```
public static void showMatrix(int[][] a)
{
    int row, column;
    for (row = 0; row < a.length; row++)
    {
        for (column = 0; column < a[row].length; column++)
            System.out.print(a[row][column] + " ");
        System.out.println();
    }
}
```

Multidimensional Arrays

- Returning Multi- Dimensional Array:

/**

Precondition: Each dimension of a is at least the value of size.

The array returned is the same as the size-by-size upper upper-left corner of the array a.

*/

```
public static double[][] corner(double[][] a, int size)
{
    double[][] temp = new double[size][size];
    int row, column;
    for (row = 0; row < size; row++)
        for (column = 0; column < size; column++)
            temp[row][column] = a[row][column];
    return temp;
}
```