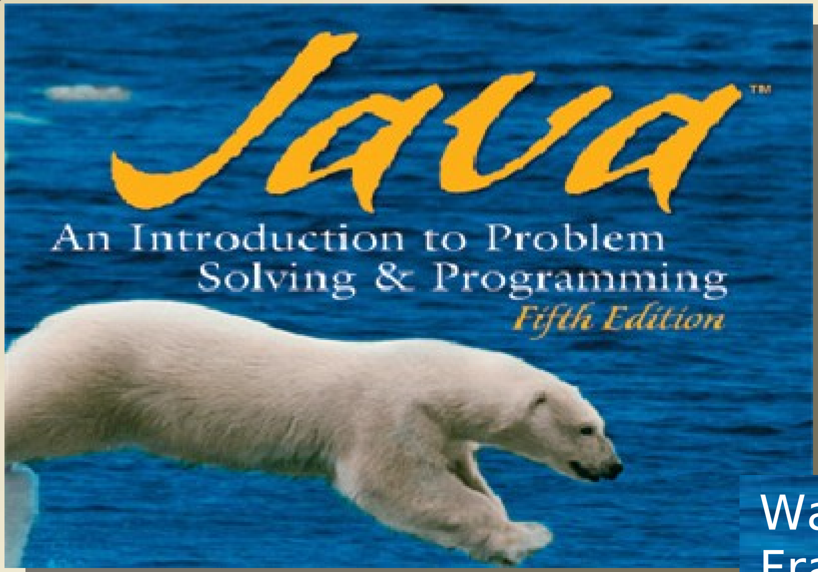


Walter Savitch
Frank M. Carrano

# Arrays

Chapter 7

## Array Basics: Outline

Creating and Accessing Arrays

Array Details

The Instance Variable length

More About Array Indices

Partially-filled Arrays

Working with Arrays

## Creating Arrays

An array is a special kind of object

Think of it as collection of variables of same type

Creating an array with 7 variables of type double

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

## Accessing Arrays

To access an element use

the name of the array

an index number enclosed in braces

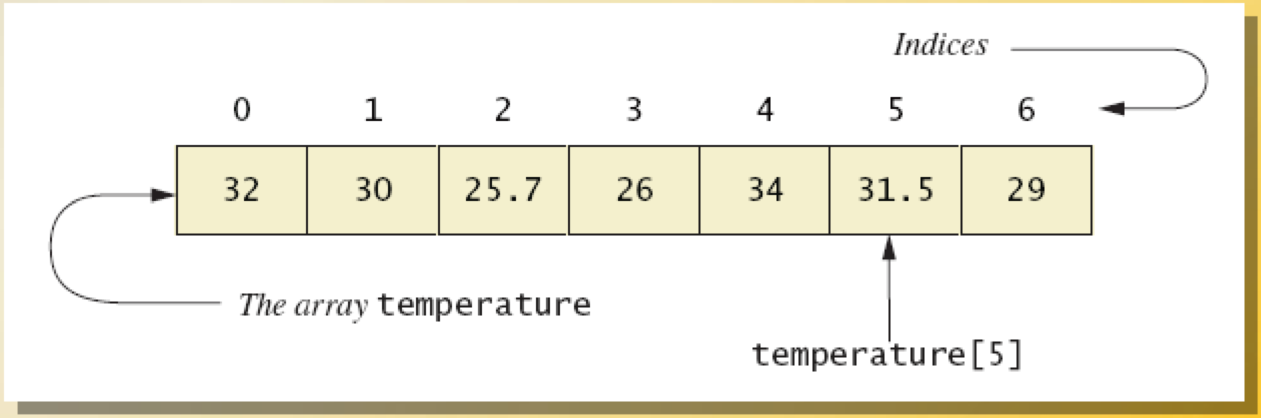
Array indices begin at zero

Example:

```
double[] temperature = new double[7];
temperature[0] = 25;
temperature[1] = 18;
```

# Creating and Accessing Arrays

Figure 7.1 A common way to visualize an array



Download and run ch07 **`ArrayOfTemperatures`**

# Array Details

Syntax for declaring an array with **new**

```
Base_Type[] Array_Name = new Base_Type[Length];
```

The number of elements in an array is its length

The type of the array elements is the array's base type

## Square Brackets with Arrays

With a data type when declaring an array

```
int[] pressure;
```

Enclose an integer expression to declare the length of the array

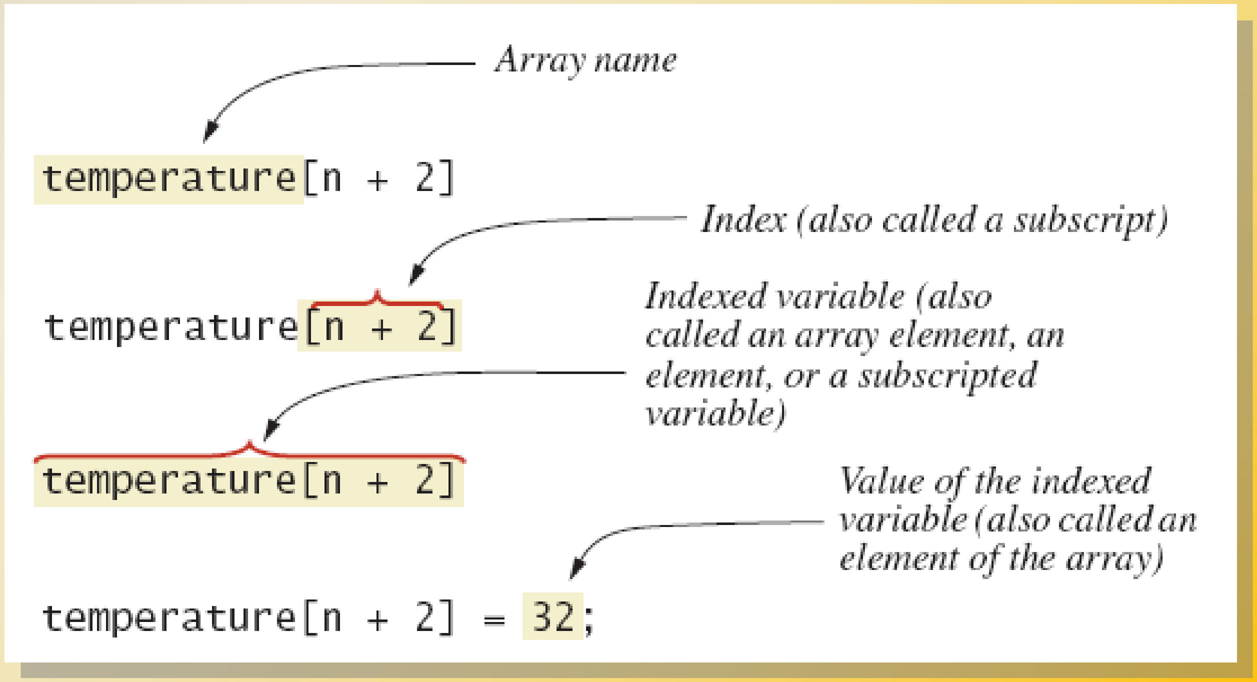
```
pressure = new int [100];
```

Naming an indexed value of the array

```
pressure[3] = keyboard.nextInt();
```

## Array Details

Figure 7.2 Array terminology


Array name
temperature[n + 2]
Index (also called a subscript)
temperature[n + 2]
Indexed variable (also called an array element, an element, or a subscripted variable)
temperature[n + 2]
Value of the indexed variable (also called an element of the array)
temperature[n + 2] = 32;

## Exercise

Write a program called `ArrayStuff` that declares an array of Strings called `friends` with length 4 and fills it with names of your friends

Use a for-loop to print all 4 elements of your array

## The Instance Variable `length`

As an object an array has only one public instance variable

Variable `length`

Contains number of elements in the array

It is final, i.e. its value cannot be changed

Download and run ch07 `ArrayOfTemperatures2`

## More About Array Indices

Index of first array element is 0

Last valid Index is `arrayName.length – 1`

Array indices must be within bounds to be valid

When program tries to access outside bounds, a run time error occurs

## Exercise

Modify your ArrayStuff program to use `length` instead of 4

## Initializing Arrays

Possible to initialize at declaration time

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

Also may use normal assignment statements

One at a time

In a loop

```
int[] count = new int[100];
for (int i = 0; i < 100; i++)
    count[i] = 0;
```

## Indexed Variables as Method Arguments

Indexed variable of an array

Example … `a[i]`

Can be used anywhere a variable of array base type can be used

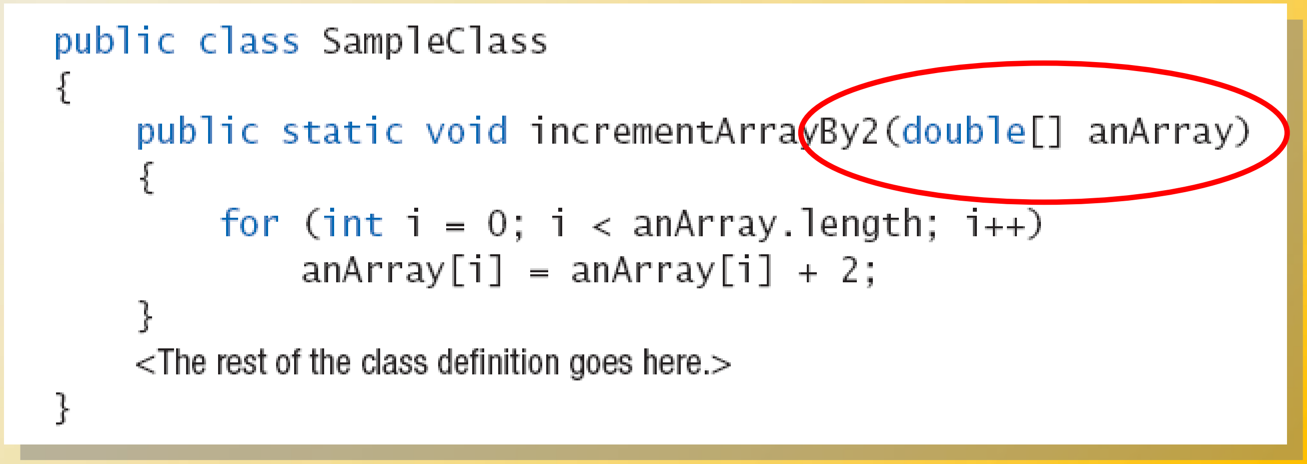
**Exercise:**

Print only those names in your friends array that are more than 5 characters long

# Entire Arrays as Arguments

Declaration of array parameter is similar to how an array is declared

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.>
}
```

## Entire Arrays as Arguments

Note: An array parameter in a method heading does not specify the length

An array of any length can be passed to the method

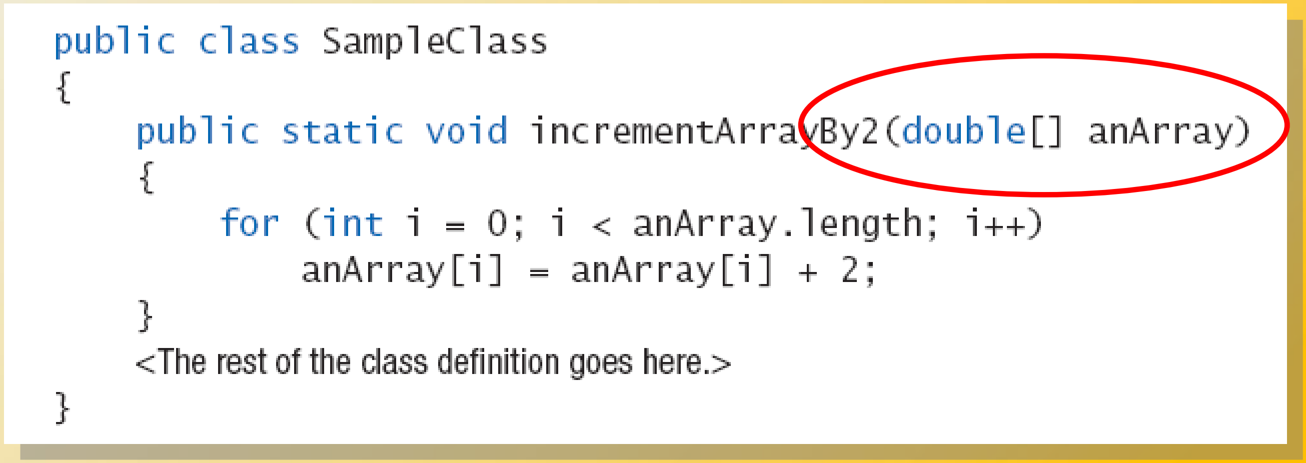
Inside the method, elements of the array can be changed

When you pass the entire array, do not use square brackets in the actual argument

## Exercise

Add a similar method to ArrayStuff called `printArray` that takes an array of Strings and prints each element.

Use `printArray` to print `friends`



```
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.>
}
```

## Arguments for Method main

Recall heading of method **`main`**

```
public static void main (String[] args)
```

This declares an array

Formal parameter named **`args`**

Its base type is **`String`**

Thus it is possible to pass to the run of a program multiple strings

These can then be used by the program

## Exercises

Ex1:

Call your printArray method with `args`

In the interactions pane, type

java ArrayStuff hello world

Ex2:

Write a program called `Adder` that adds all of the numbers in `args` and print the result

use Double.parseDouble(String)

In the interactions pane:

java Adder 1 2 3 4 5

## Array Assignment and Equality

Arrays are objects

Assignment and equality operators behave (misbehave) as specified in previous chapters

Variable for the array object contains memory address of the object

Assignment operator **=** copies this address

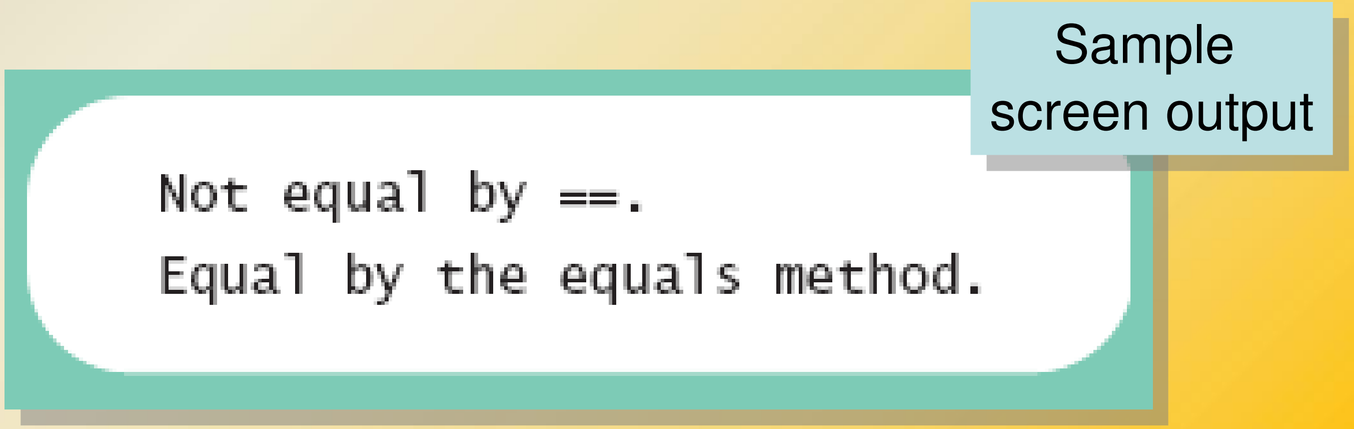
Equality operator **==** tests whether two arrays are stored in same place in memory

## Array Assignment and Equality

Two kinds of equality

View example program, listing 7.6

**`class TestEquals`**


Sample screen output

```
Not equal by ==.
Equal by the equals method.
```

## Array Assignment and Equality

Note results of `==`

Note definition and use of method `equals`

Receives two array parameters

Checks length and each individual pair of array elements

Remember: Array types are reference types

## Methods that Return Arrays

A Java method may return an array

```
public static int[] add5(int[] anArray)
```

Note definition of return type as an array

To return the array value

Declare a local array

Use that identifier in the `return` statement

## Exercise

Add a method copyArray to ArrayStuff:

```
public static String[] copyArray(String[] anArray)
{
    // declare array to return
    // copy anArray to return array
    // return the copied array
}
```

## Partially Filled Arrays

Array size specified at creation

can't be changed after that

Some elements of the array might be empty

This is termed a *partially filled array*

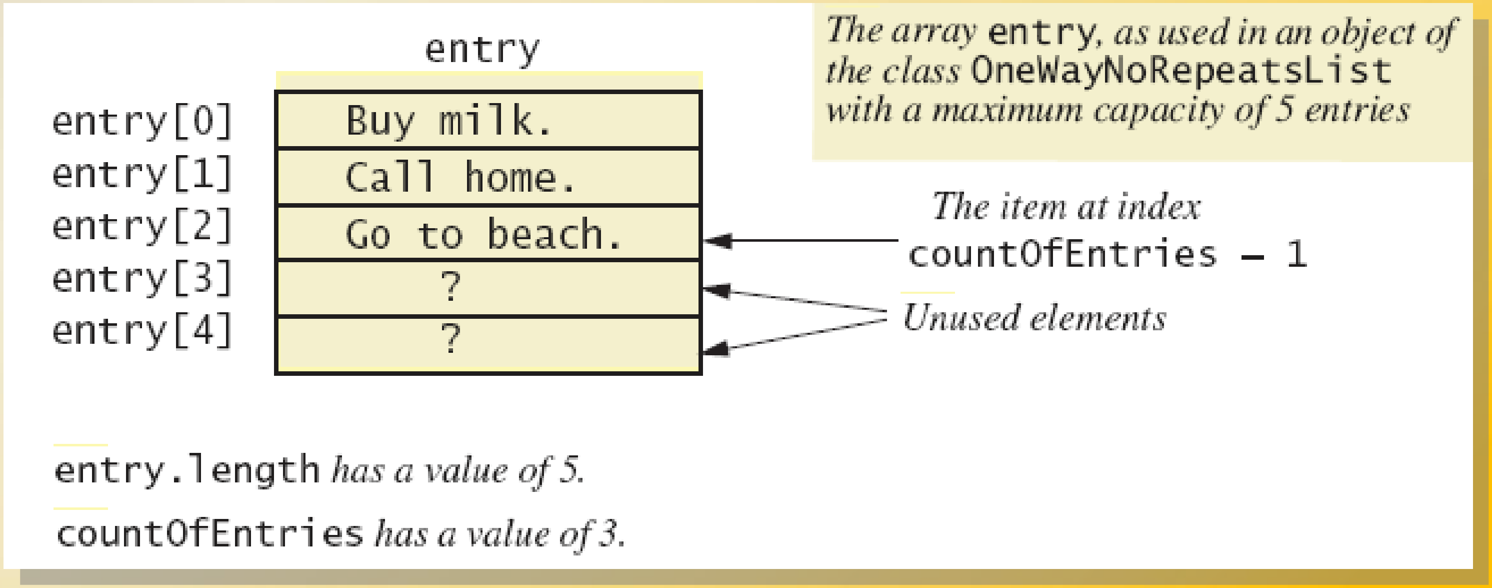
Programmer must keep track of how much of array is used

## Partially Filled Arrays

Download from Examples link:

StringList.java

StringListDemo.java


entry
entry[0]
entry[1]
entry[2]
entry[3]
entry[4]
Buy milk.
Call home.
Go to beach.
?
?
The array entry, as used in an object of the class OneWayNoRepeatsList with a maximum capacity of 5 entries
The item at index countOfEntries – 1
Unused elements
entry.length has a value of 5.
countOfEntries has a value of 3.

## Searching an Array

Algorithm used in the StringList.contains method is called **sequential search**

Looks in order from first to last

Good for unsorted arrays

Search ends when

Item is found … or …

End of list is reached

If list is sorted, we could use a more efficient search method

## Working with Arrays

## Common Tasks

When working with arrays, there are some operations that need to be performed in many situations.

These operations include:

printing

copying

resizing

removing an element

inserting an element

## Printing an Array

Unlike other objects, there is no simple one-liner for printing an entire array.

You will have to code it using a for-loop:

```
int[] myArray = {4, 6, 2, 3, 7};
for (int i=0; i < myArray.length; i++)
{
    System.out.print(myArray[i] + " ");
}
System.out.println();
```

## Printing an Array

To print a partially-filled array containing `numElements`:

```
int[] myArray;
// partially fill array - increment
// numElements each time an element is
// added
for (int i=0; i < numElements; i++)
{
    System.out.print(myArray[i] + “ “);
}
System.out.println();
```

## Copying an Array

Pseudocode:

create a new array of the same length

copy each element from `myArray` to the new array

```
int[] result = new int[myArray.length];
for (int i=0; i < myArray.length; i++)
{
     result[i] = myArray[i];
}
```

## Copying an Array

To make a copy of only the filled part of the partially-filled array `myArray`, where `numElements` are filled:

```
int[] result = new int[____________];
for (int i=0; i < ____________; i++)
{
     result[i] = myArray[i];
}
```

## Copying an Array

To make a copy of only the filled part of the partially-filled array `myArray`, where `numElements` are filled:

```
int[] result = new int[numElements];
for (int i=0; i < numElements; i++)
{
     result[i] = myArray[i];
}
```

## Copying an Array with System.arraycopy

Entire array:

```
int[] result = new int[myArray.length];
System.arraycopy(myArray, 0, result, 0,
          myArray.length);
```

First `numElements` elements:

```
int[] result = new int[numElements];
System.arraycopy(myArray, 0, result, 0,
          numElements);
```

## Resizing an Array

When you need to add another element to a full array, **resize** it.

Resizing just means making the array bigger by some amount.

## Resizing an Array

Pseudocode:

```
create a new array amount larger than myArray
copy all elements from myArray to new array
make myArray  reference the new array
```

```
int[] result = new int[myArray.length + amount];
for (int i=0; i < myArray.length; i++)
{
     result[i] = myArray[i];
}
myArray = result;
```

## Resizing an Array with System.arraycopy

```
int[] result = new int[myArray.length + amount];
System.arraycopy(myArray, 0,result, 0,
                 myArray.length);
```

## Removing an Element from an Array

Pseudocode:

```
create a new array 1 smaller than myArray
copy before index from myArray to new array
copy elements after index to new array (at index-1)
make myArray reference the new array
```

## Removing an Element from an Array

Fill in the blanks:

```
int[] result = new int[myArray.length-1];
// copy elements before index
for (int i=0; i < index; i++)
{
    result[i] = myArray[i];
}
// copy elements after index
for (int i=index+1; i < myArray.length; i++)
{
    result[____] = myArray[___];
}
myArray = result;
```

# Removing an Element from an Array

```
int[] result = new int[myArray.length-1];
// copy elements before index
for (int i=0; i < index; i++)
{
    result[i] = myArray[i];
}
// copy elements after index
for (int i=index+1; i < myArray.length; i++)
{
    result[i-1] = myArray[i];
}
myArray = result;
```

## Removing an Element from an Array with System.arraycopy

```
int[] result = new int[myArray.length-1];
System.arraycopy(myArray, 0, result, 0, index);
System.arraycopy(myArray,index+1,
                 result, index,
                 myArray.length-1-index);
myArray = result;
```

## Inserting an Element into an Array

Pseudocode:

```
create a new array 1 bigger than myArray
copy elements before index from myArray to new array
insert element at index
copy elements after index to new array (at index+1)
make myArray reference the new array
```

# Inserting an Element into an Array

Fill in the blanks:

```
int[] result = new int[myArray.length+1];
// copy elements before index
for (int i=0; i < index; i++)
{
    result[i] = myArray[i];
}

result[index] = elementToInsert;
// copy elements after index
for (int i=index; i < myArray.length; i++)
{
    result[____] = myArray[___];
}
myArray = result;
```

## Inserting an Element into an Array

```
int[] result = new int[myArray.length+1];
// copy elements before index
for (int i=0; i < index; i++)
{
    result[i] = myArray[i];
}

result[index] = elementToInsert;
// copy elements after index
for (int i=index; i < myArray.length; i++)
{
    result[i+1] = myArray[i];
}
myArray = result;
```

## Inserting an Element into an Array with System.arraycopy

```
int[] result = new int[myArray.length+1];
// copy elements before index
System.arraycopy(myArray, 0, result, 0, index);

result[index] = element;

// copy elements after index
System.arraycopy(myArray, index,
                 result, index+1,
                 myArray.length-index);
myArray = result;
```

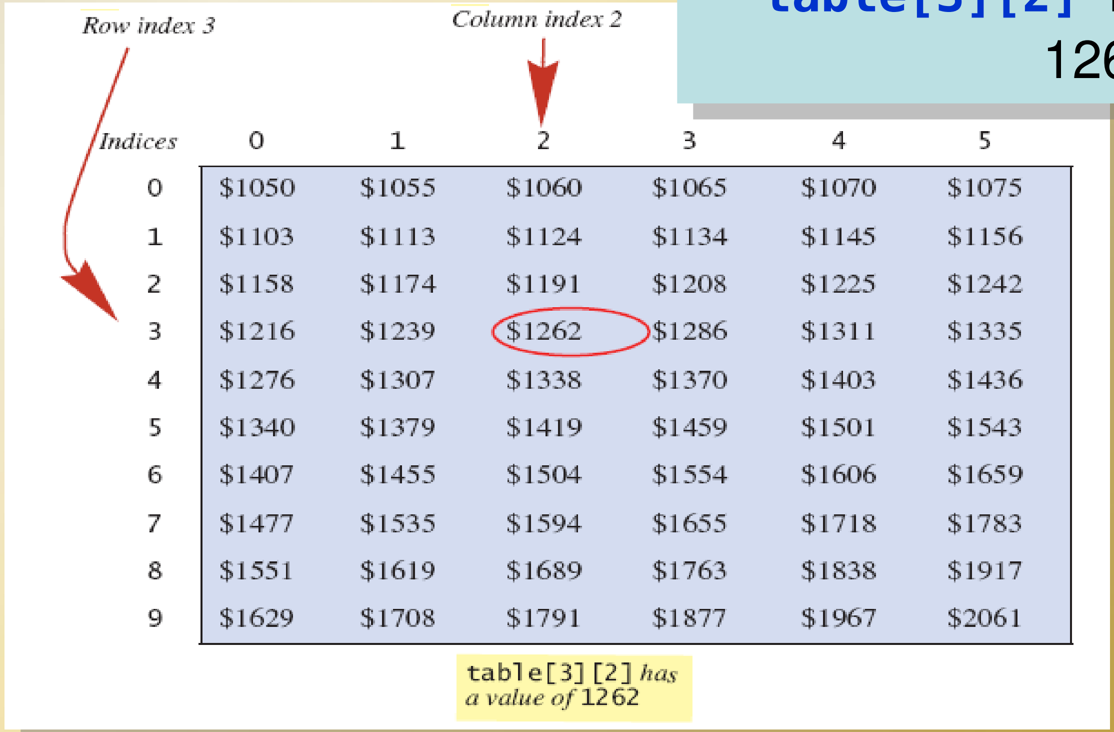
## Multidimensional Arrays: Outline

- Multidimensional-Array Basics
- Multidimensional-Array Parameters and Returned Values
- Java's Representation of Multidimensional Arrays
- Ragged Arrays
- Programming Example: Employee Time Records

# Multidimensional-Array Basics

Figure 7.7 Row and column indices for an array named **`table`**

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



Row index 3
Column index 2

| 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-Array Basics

We can access elements of the table with a nested for loop

Example:

```
for (int row = 0; row < 10; row++)
   for (int column = 0; column < 6; column++)
      table[row][column] =
          balance(1000.00, row + 1, (5 + 0.5 * column));
```

## Multidimensional-Array Parameters and Returned Values

Methods can have

Parameters that are multidimensional-arrays

```
public static void printTable(int[][] table)
{
...
}
```

Return values that are multidimensional-arrays

```
public static int[][] copyTable(int[][] table)
{
...
}
```

## Java's Representation of Multidimensional Arrays

Multidimensional array represented as several one-dimensional arrays

Given

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

Array **table** is actually a 1 dimensional array of length 10, with base type `int[]`

It is an array of arrays.

Important when sequencing through multidimensional array

## Ragged Arrays

Not necessary for all rows to be of the same length

Example:

```
int[][] b;
b = new int[3][];
b[0] = new int[5]; //First row,  5 elements
b[1] = new int[7]; //Second row, 7 elements
b[2] = new int[4]; //Third row,  4 elements
```

## Printing 2D Arrays

Use table.length and table[row].length

Outer loop iterates the rows

Inner loop iterates columns in current row

```
public static void printArray(int[][] table)
{
  for (int row=0; row < table.length; row++)
  {
    for (int col=0; col < table[row].length; col++)
    {
      System.out.print(table[row][col]);
    }
    System.out.println();
  }
}
```