Arrays

Chapter 9
Chapter Objectives

- Investigate Java one-dimensional arrays
- Study sorting and searching of lists
- Implement command-line arguments
- Introduce two-dimensional arrays
- Build a matrix class
- Show use of arrays in graphical programming
- Describe use of matrix methods for solving linear systems
Example: Ms. White's Test Score Analyzer

Problem:
Ms. White needs to analyze students' test performance. She needs a program to display a name and let her enter the test score. Then it should compute and display the average. Finally it should give a report with names, scores, and deviation from the average.
Test Analysis -- enter scores:
Aardvark, James : 92
Biffkirk, Sue : 79
Crouse, Nancy : 95
...  
Average = 87.39

Summary . . .
# Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Kind</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence of names</td>
<td>constant</td>
<td>String[]</td>
<td>STUDENTS</td>
</tr>
<tr>
<td>each name</td>
<td>varying</td>
<td>String</td>
<td>STUDENTS[i]</td>
</tr>
<tr>
<td>sequence of scores</td>
<td>varying</td>
<td>double[]</td>
<td>scores</td>
</tr>
<tr>
<td>each score</td>
<td>varying</td>
<td>double</td>
<td>scores[i]</td>
</tr>
<tr>
<td>the screen</td>
<td>varying</td>
<td>Screen</td>
<td>theScreen</td>
</tr>
<tr>
<td>prompt for each score</td>
<td>varying</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>average</td>
<td>varying</td>
<td>double</td>
<td>average</td>
</tr>
</tbody>
</table>
Algorithm

1. Define **STUDENTS** array to hold names, array **scores** to hold test scores
2. For each student in array
   a) display name & prompt
   b) read double, store in array scores
3. Compute **average**, display it
4. For each student in array
   a) Display name, test score, difference between that score and **average**
Coding and Testing

Note source code, Figure 9.1

Features:

- array of student names initialized at declaration
- use of `NUMBER_OF_STUDENTS` constant
- for loops to process the arrays
9.2 Arrays

Java arrays are objects
must be accessed via handles

Defining an array

```
Type [] name
```

- This declares the handle only
- Initialized to null
- Stores an address when arrays are created

Where:

- `Type` specifies the kind of values the array stores
- the brackets `[]` indicate this is an array
- `name` is the handle to access the array
Definitions Using Array Literals

Used when exact size and initial values of an array are known in advance

used to initialize the array handle

```c
int [] count = { 0,0,0,0}
```

Visualize the results of the above command

```
Count
[0] [1] [2] [3]
0 0 0 0
```
Consider student names in Section 9.1

```java
final String[] STUDENTS = {
  "Bashful", "Doc", ..., "Sneezy"
};
```

Note results:
Contrast Definitions

Elements are primitive values (0s in this case)

Elements are handles for Strings
Definitions using `new`:

double [] scores = new double [NUMBER_OF_STUDENTS];

- The above statement will:
  - allocate block of memory to hold 7 doubles
  - initialize block with zeros
  - return the address of the block
  - create a handle called `scores`
  - store address of the block in `scores`
Syntax

Forms

```
ElementType [] arrayName;
ElementType [] arrayName = new ElementType [size];
ElementType [] arrayName = array-literal;
```

Where:

- `ElementType` is any type
- `arrayName` is the handle for the array
- `array-literal` is a list of literals enclosed in curly braces `{   }`
Access individual elements of an array using:
- the name (handle) of the array
- a number (index or subscript) that tells which of the element of the array

What value is stored in `count[2]`?
Often a `for()` loop is used to process each of the elements of the array in turn.

```java
for (int i = 0; i < NUM_STUDENTS; i++)
{
    theScreen.print( STUDENTS[i] + "\t"+scores[i]+"\t" );
    theScreen.println(scores[i]-average);
}
```

The loop control variable, `i`, is used as the index to access array components.
Array Parameters

- Methods can accept arrays via parameters
- Use square brackets [ ] in the parameter declaration:

```java
public static double sum(double[] array)
{
    for (int i=0; i < array.length; i++)
    {
        // ... 
    }
}
```

- This works for any size array
- Use the .length attribute of an array to control the for loop
Methods that Return Array Values

A method may return an array as its result.

```java
public static double[] readArray() {
  // ask for now many, n
  double result[] = new double[n];
  for (i = 0; i < n; i++) {
    theScreen.print("Enter ...");
    result[i] = theKeyboard.readDouble();
  }
  return result;
}
```

Declare return type

Declare local array for receiving input

Local array is returned
Java provides a few operations to use with arrays, including assignment

Consider:

```java
int [] alist = { 11, 12, 13, 14};
int [] blist;
blist = alist;
```

Recall that `alist` and `blist` are handles

- `alist` contains an address of where the numbers are
- `blist` now contains that same address
- `blist` does not have a copy of `alist`
Array Cloning

To actually create another array with its own values, Java provides the `.clone()` method

```java
int [] alist = { 11, 12, 13, 14};
int [] blist;
blist = alist.clone();
```

Now there are two separate lists of numbers, one with handle `alist`, the other with handle `blist`
Recall previous declaration:

```
String[] s_list = STUDENTS.clone();
```

This will create another list of handles, also pointing to the names.

Called a “shallow copy” operation.

Elements are handles for String values.
We can write our own “deep copy” method

```java
public String[][] deepCopy(String[][] original) {
    String[][] result =
        new String[original.length][];
    for (i = 0; i < original.length; i++)
        result[i] = original[i].clone();
    return result;
}
```
Array Equality

Java has an `equals()` method for classes

```java
if (a1.equals(a2)) ...
```

If `a1` and `a2` are arrays, the `equals()` method just looks at the addresses the handles point to.

They could be pointing to different addresses but the contents of the array still be equal.

It is the contents that we really wish to compare.
Array Equality

We must write our own method to compare the arrays.

- They both must have the same length.
- Then use a `for()` loop to compare element by element for equality.

```java
if (list1.length == list2.length)
{
    for (int i = 0; i < list1.length; i++)
        if (list1[i] != list2[i])
            return false;
    return true;
} else
    return false;
```

Method returns as soon as one pair is not equal.
Arrays are fixed in size at declaration time

cannot grow or shrink dynamically during run time

Java provides a Vector class which can grow or shrink during the run of the program

Note methods provided, Table 9.1 from text
9.3 Sorting

Arranging items in a list so they are in either ascending or descending order

There are several algorithms to accomplish this – some are known as:

- selection sort
- linear insertion sort
- quicksort
Selection Sort

- Make passes through the list (or part of it)
  - on each pass select one item (smallest or largest) to be correctly positioned
  - exchange that item with the one that was in its place

```
67 33 21 84 49 50 78
21 33 67 84 49 50 78
```
Selection Sort

- On successive passes, must only check beyond where smallest item has been placed.

- A pair of nested `for()` loops will accomplish this checking and swapping of values.

- Note: this is a simple but inefficient algorithm.
Linear Insertion Sort

- As list is built, keep it sorted
- place new values in the list where they belong

Tasks include
- finding where the new item belongs
- shifting all elements from that slot and beyond over one slot
- inserting the new item in the vacated spot

Again, a double nested loop can accomplish this task

This is efficient for small lists
Quicksort

- Very efficient
- Implemented by a recursive algorithm

Strategy
- choose a pivot point
- perform exchanges so all elements to the left are less than the pivot
- all elements on the right are greater
- this makes 2 sub lists where the same strategy with the pivot can be performed (repeatedly)
- this is a "divide and conquer" concept
9.4 Searching

- **Linear search**
  - begin with first item in a list
  - search sequentially until desired item found or reach end of list

- **Binary search (of a sorted list)**
  - look at middle element
  - if target value found, done
  - if target value is higher, look in middle of top half of list
  - if target value is lower, look in middle of lower half of list
Contrast Search Algorithms

- Linear search
  - with n items in the list, may require n comparisons to find target

- Binary search
  - with n items in the list, requires \( \text{at most} \ \log_2 n \) comparisons
  - for example with 1024 items, maximum of 10 comparisons

Conclusion: Binary search is more efficient
9.5 Processing Command-Line Arguments

Recall the heading for the method `main()`
```
public static void main(String[] args)
{
    . . .
}
```

From this chapter we know that `args` is a handle for an array of `String` values

These string values can be used in the program
In UNIX and MS–DOS the user types in commands at the prompt:

```
C:\> mkdir mydocs
```

- `mkdir` is a program
- `mydocs` is the argument sent to the program

This is where we can use `args`
Running Java from the Command Line

- At the command line enter:
  ```
  java ClassName
  ```
- The `java` interpreter is called. `ClassName` is an argument for that program.
- It would also be possible to enter
  ```
  java ClassName argmt₁ argmt₂ ... argmtₙ
  ```
- The interpreter builds a `String` array of `n` elements for `ClassName`.
- `arg[i]` is the handle for `argmtᵢ`. 
Using Command-Line Arguments

Consider source code that would list the command line arguments received by a program:

```
System.out.println( "There are " + args.length+" arguments");
for (int i =0; i < args.length; i++)
  System.out.println( args[i]);
```
Example: Square Root Calculator

- The program will be run from the command-line.
- Given a real value (or values) it will print the square root for all values received as command-line arguments.

```
java Sqrt 4 25 2
2
5
1.414
```
## Objects

<table>
<thead>
<tr>
<th>Object</th>
<th>Kind</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence of arguments</td>
<td>variable</td>
<td>String[]</td>
<td>args</td>
</tr>
<tr>
<td>number of arguments</td>
<td>variable</td>
<td>int</td>
<td>args.length</td>
</tr>
<tr>
<td>particular argument</td>
<td>variable</td>
<td>double</td>
<td>inValue</td>
</tr>
<tr>
<td>square root of the argument</td>
<td>variable</td>
<td>double</td>
<td></td>
</tr>
</tbody>
</table>
Algorithm

1. If `args.length < 1`, display error message, quit

2. For each integer `i`, 0 to `args.length - 1`...
   a) change the `String` value, `arg[i]`, to a double, `inValue`
   b) display `inValue` and its square root

Note source code Figure 9.3 in text
Arrays studied so far have one dimension ... length
It is also possible to define arrays with more than one dimension
Two dimensional arrays can store values arranged in rows and columns
Three dimensional arrays can store values arranged in rows, columns, and ranks (or pages)
Sample Problem:

Consider a trucking business with centers in 6 cities. The owner requires a computerized mileage chart for the drivers.

Given any of the two cities, the program must display the approximate mileage between them.
Program Behavior

- Display a numbered menu of cities
- Read the numbers of two cities from keyboard
- Look up mileage between the cities in a two dimensional table of values
- Display the mileage
## Objects

<table>
<thead>
<tr>
<th>Objects</th>
<th>Kind</th>
<th>Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>menu of cities</td>
<td>constant</td>
<td>String</td>
<td>CITY_MENU</td>
</tr>
<tr>
<td>number of 1st city</td>
<td>varying</td>
<td>int</td>
<td>city1</td>
</tr>
<tr>
<td>number of 2nd city</td>
<td>varying</td>
<td>int</td>
<td>city2</td>
</tr>
<tr>
<td>mileage chart</td>
<td>constant</td>
<td>int[][]</td>
<td>MILEAGE_CHART</td>
</tr>
<tr>
<td>the mileage</td>
<td>varying</td>
<td>int</td>
<td>mileage</td>
</tr>
</tbody>
</table>
Algorithm

- Define
  - `MILEAGE_CHART` (2D array)
  - `CITY_MENU` (list of supported cities)
- Display `CITY_MENU`
- Read two integers from keyboard into `city1` and `city2`
- Calculate `mileage` by looking up `MILEAGE_CHART[city1][city2]`
- Display `mileage`
Note source code, Figure 9.4

accompanying sample run

Note definition and initialization of

```java
final int[][][] MILEAGE_CHART = {
    { 0, 97 ... 130 }, // Daytona
    { 97, ... 128 },
    ...,
    { 130 ... 0 } }; // Tampa
```
Accessing 2D Array Elements

- Requires two indices
- Which cell is `MILEAGE_CHART[3][2]`?

<table>
<thead>
<tr>
<th></th>
<th>[0]</th>
<th>[1]</th>
<th>[2]</th>
<th>[3]</th>
<th>[4]</th>
<th>[5]</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0]</td>
<td>0</td>
<td>97</td>
<td>90</td>
<td>268</td>
<td>262</td>
<td>130</td>
</tr>
<tr>
<td>[1]</td>
<td>97</td>
<td>0</td>
<td>74</td>
<td>337</td>
<td>144</td>
<td>128</td>
</tr>
<tr>
<td>[2]</td>
<td>90</td>
<td>74</td>
<td>0</td>
<td>354</td>
<td>174</td>
<td>201</td>
</tr>
<tr>
<td>[3]</td>
<td>268</td>
<td>337</td>
<td>354</td>
<td>0</td>
<td>475</td>
<td>269</td>
</tr>
<tr>
<td>[4]</td>
<td>262</td>
<td>144</td>
<td>174</td>
<td>475</td>
<td>0</td>
<td>238</td>
</tr>
<tr>
<td>[5]</td>
<td>130</td>
<td>128</td>
<td>201</td>
<td>269</td>
<td>238</td>
<td>0</td>
</tr>
</tbody>
</table>
Defining 2D Array Operations

- As with 1D arrays, variables and constants are handles to array objects.
- Take care with assignment operator.
  - use `clone()` method for shallow copy.
  - write our own method for deep copy.
- Recall that `equals()` compares handles of the 2D arrays.
  - we must define our own `equals()` method for comparing individual values.
Matrix Class

A two-dimensional array with m rows and n columns is called an \( m \times n \) matrix.

Mathematicians perform a variety of operations:
- matrix multiplication
- matrix addition, subtraction
Matrix Multiplication Algorithm

1. If no. columns in mat1 ≠ no. rows in mat2, product not defined, terminate

2. For each row i in mat1
   For each column j in mat2
     a) set sum = 0
     b) for each column k in mat1 add mat1[i][k]*mat2[k][j] to sum
     c) set mat3[i][j] equal to sum
Building a **Matrix** Class

- Operations needed
- Default constructor, initialize empty
- Explicit value constructor, specify size
- Input: fill with values from keyboard
- Output: convert to **String** representation
- Retrieve element at \([r][c]\)
- Change value at \([r][c]\)
- Return product of this matrix with another
public class Matrix {
    public Matrix() { } // stub for constructor
    public Matrix (int rows, int columns) { } // explicit constructor
    // multiply method

    private int myRows;
    private int myColumns;
    private double [][] myArray;
}
public Matrix()
{
    myArray = null;
    myRows = 0;
    myColumns = 0;
}

Usage:
Matrix m = new Matrix();
Explicit-Value Constructor

```java
public Matrix(int rows, int columns) {
    if (rows < 0 || columns < 0)
        ...// error message
    else
    {
        myArray = new double[rows][columns];
        myRows = rows;
        myColumns = columns;
    }
}
```

Usage:
```java
Matrix m = new Matrix (7,12);
```
Other Methods

- Matrix Output, `toString()`, Figure 9.7
- Matrix Input, `read()`, Figure 9.8
- Attribute Accessors, Figure 9.9
  - `return myRows`
  - `return myColumns`
- Individual element accessor, mutator Figure 9.10
- Multiplication method Figure 9.11
Array Declaration Syntax

Form

\[ \text{Type} \ [\] \ [] \ldots \ [] \ \text{identifier} = \]
\[ \text{new Type}[\text{DIM}_1][\text{DIM}_2] \ldots [\text{DIM}_n]; \]

Where

- **Type** is any known type
- **identifier** is the handle for the array
- each **DIM\_i** is an expression that evaluates to a non-negative integer
Problem: Profit Analysis for Middle Earth Touring Company

Program will prompt for proceeds on different tours

Then display pie chart showing relative percentage of profit for each tour

Program should be upgradeable for adding more tours in the future
After entries are made for each tour destination, the program displays a window with a pie chart and legend with color codes and percentages for tour names.
Operations for Pie-Chart Class

- Construct a pie chart given
  - String array with slice labels
  - Double array containing raw values for slices
- Draw (paint) the chart
- Change color of given slice
- Change raw value of given slice
- We must also determine attributes for the program – window dimensions, etc.
PieChartPanel

Constructor

Receive as inputs

- SLICE_LABELS, a constant String array
- sliceValues, a double array

Use a utility method
computePercentages() to initialize attribute MySlicePercent
paintComponent() Method

- **Swing** components expect a method with this name
- Use `fillArc()` from the Graphics class
- Use `fillRect()` to draw legend box
- Use `drawString()` to label legend box
- Note the care taken to get arguments correct (location and size) for these methods
- Slices drawn by computing starting angle and size of angle based on %
Other Methods

- **setSliceColor()**
  - Receives slice number and new color

- **setSliceValue()**
  - Receives slice number
  - New value

Note full source code, Figure 9.14
Upgrading the Class

- Design of class enables ease of upgrading
- Only changes are including additional tour names in the TOURS array
- Note changes in Figure 9.15
- View output of sample run
Application of Matrices: Solving linear systems

\[ 3x_1 + 4x_2 = 7 \]
\[ 4x_1 - 8x_2 = 12 \]

Find the values (if any) for \( x_1 \) and \( x_2 \) that make the above pair of equations true statements.

Strategy used is called Gaussian elimination.

Manipulate the matrix to solve the system.