Chapter 7

Arrays
Chapter Scope

• Array declaration and use
• Bounds checking
• Arrays as objects
• Arrays of objects
• Command-line arguments
• Variable-length parameter lists
• Multidimensional arrays
Arrays

- An **array** is an ordered list of values

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
<td>87</td>
<td>94</td>
<td>82</td>
<td>67</td>
<td>98</td>
<td>87</td>
<td>81</td>
<td>74</td>
<td>91</td>
</tr>
</tbody>
</table>

The entire array has a single name. Each value has a numeric *index*.

An array of size N is indexed from zero to N-1.

This array holds 10 values that are indexed from 0 to 9.
Arrays

• A particular value in an array is referenced using the array name followed by the index in brackets.

• For example, the expression

\[ \text{scores}[2] \]

refers to the value 94 (the 3rd value in the array).

• That expression represents a place to store a single integer and can be used wherever an integer variable can be used.
Arrays

- Arrays can be depicted vertically or horizontally

![Diagram of an array with indices and values]
Arrays

• An array element can be assigned a value, printed, or used in a calculation

```java
scores[2] = 89;
scores[first] = scores[first] + 2;
mean = (scores[0] + scores[1])/2;
System.out.println("Top = " + scores[5]);
```
Arrays

• The values held in an array are called *array elements*

• An array stores multiple values of the same type – the *element type*

• The element type can be a primitive type or an object reference

• Therefore, we can create an array of integers, an array of characters, an array of *String* objects, an array of *Coin* objects, etc.

• In Java, the array itself is an object that must be instantiated
Declaring Arrays

• The scores array could be declared as follows

  \[
  \text{int}\[\] \text{scores} = \text{new int}[10];
  \]

• The type of the variable scores is int[] (an array of integers)

• Note that the array type does not specify its size, but each object of that type has a specific size

• The reference variable scores is set to a new array object that can hold 10 integers
Declaring Arrays

Creating an Array

type of the array
(no size)
creates new
array object

```
double[] discounts = new double[35];
```

array name
type and size
Declaring Arrays

• Some other examples of array declarations

```java
float[] prices = new float[500];

boolean[] flags;
flags = new boolean[20];

char[] codes = new char[1750];
```
Using Arrays

• The for-each loop can be used when processing array elements:

```java
for (int score : scores)
    System.out.println(score);
```

• This is only appropriate when processing all array elements from the lowest index to the highest index
public class BasicArray
{
  //-----------------------------------------------------------------
  //  Creates an array, fills it with various integer values,
  //  modifies one value, then prints them out.
  //-----------------------------------------------------------------
  public static void main(String[] args)
  {
    final int LIMIT = 15, MULTIPLE = 10;

    int[] list = new int[LIMIT];

    // Initialize the array values
    for (int index = 0; index < LIMIT; index++)
      list[index] = index * MULTIPLE;

    list[5] = 999;  // change one array value

    // Print the array values
    for (int value : list)
      System.out.print(value + "  ");
  }
}
BasicArray Example

The array is created with 15 elements, indexed from 0 to 14.

After three iterations of the first loop:
- Index 0: 0
- Index 1: 10
- Index 2: 20
- Index 3: 30
- Index 4: 40
- Index 5: 50
- Index 6: 60
- Index 7: 70
- Index 8: 80
- Index 9: 90
- Index 10: 100
- Index 11: 110
- Index 12: 120
- Index 13: 130
- Index 14: 140

After completing the first loop:
- Index 0: 0
- Index 1: 10
- Index 2: 20
- Index 3: 30
- Index 4: 40
- Index 5: 50
- Index 6: 60
- Index 7: 70
- Index 8: 80
- Index 9: 90
- Index 10: 100
- Index 11: 110
- Index 12: 120
- Index 13: 130
- Index 14: 140

After changing the value of list[5]:
- Index 0: 0
- Index 1: 10
- Index 2: 20
- Index 3: 30
- Index 4: 40
- Index 5: 999
- Index 6: 60
- Index 7: 70
- Index 8: 80
- Index 9: 90
- Index 10: 100
- Index 11: 110
- Index 12: 120
- Index 13: 130
- Index 14: 140
Bounds Checking

• Once an array is created, it has a fixed size

• An index used in an array reference must specify a valid element

• That is, the index value must be in range 0 to N-1

• The Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds

• This is called automatic `bounds checking`
Bounds Checking

• For example, if the array `codes` can hold 100 values, it can be indexed using only the numbers 0 to 99.

• If the value of `count` is 100, then the following reference will cause an exception to be thrown:

  ```java
  System.out.println(codes[count]);
  ```

• It’s common to introduce off-by-one errors when using arrays:

  ```java
  for (int index=0; index <= 100; index++)
      codes[index] = index*50 + epsilon;
  ```
Bounds Checking

• Each array object has a public constant called `length` that stores the size of the array

• It is referenced using the array name

```
scores.length
```

• Note that `length` holds the number of elements, not the largest index
import java.util.Scanner;

public class ReverseOrder {
    //-----------------------------------------------------------------
    //  Reads a list of numbers from the user, storing them in an
    //  array, then prints them in the opposite order.
    //-----------------------------------------------------------------
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);

        double[] numbers = new double[10];

        System.out.println("The size of the array: " + numbers.length);

        for (int index = 0; index < numbers.length; index++) {
            System.out.print("Enter number "+(index+1)+": ");
            numbers[index] = scan.nextDouble();
        }
    }
}
System.out.println("The numbers in reverse order:");

for (int index = numbers.length-1; index >= 0; index--)
    System.out.print(numbers[index] + " ");
import java.util.Scanner;

public class LetterCount {

    // Read a sentence from the user and counts the number of 
    // uppercase and lowercase letters contained in it.
    public static void main(String[] args) {
        final int NUMCHARS = 26;

        Scanner scan = new Scanner(System.in);

        int[] upper = new int[NUMCHARS];
        int[] lower = new int[NUMCHARS];

        char current; // the current character being processed
        int other = 0; // counter for non-alphabetics

        System.out.println("Enter a sentence:");
        String line = scan.nextLine();
// Count the number of each letter occurrence
for (int ch = 0; ch < line.length(); ch++)
{
    current = line.charAt(ch);
    if (current >= 'A' && current <= 'Z')
        upper[current-'A']++;
    else
        if (current >= 'a' && current <= 'z')
            lower[current-'a']++;
        else
            other++;
}

// Print the results
System.out.println();
for (int letter=0; letter < upper.length; letter++)
{
    System.out.print((char) (letter + 'A'));
    System.out.print(": " + upper[letter]);
    System.out.print("\t\t" + (char) (letter + 'a'));
    System.out.println(": " + lower[letter]);
}

System.out.println();
System.out.println("Non-alphabetic characters: " + other);
}
Alternate Array Syntax

• The brackets of the array type can be associated with the element type or with the name of the array

• Therefore the following two declarations are equivalent

```java
float[] prices;
float prices[];
```

• The first format generally is more readable and should be used
Initializer Lists

• An *initializer list* can be used to instantiate and fill an array in one step

• The values are delimited by braces and separated by commas

• Examples:

```java
int[] units = {147, 323, 89, 933, 540, 269, 97, 114, 298, 476};

char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```
Initializer Lists

• Note that when an initializer list is used
  – the `new` operator is not used
  – no size value is specified

• The size of the array is determined by the number of items in the initializer list

• An initializer list can be used only in the array declaration
public class Primes
{
    //----------------------------------------------------------------------------
    // Stores some prime numbers in an array and prints them.
    //----------------------------------------------------------------------------
    public static void main(String[] args)
    {
        int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

        System.out.println("Array length: " + primeNums.length);

        System.out.println("The first few prime numbers are:");

        for (int prime : primeNums)
            System.out.print(prime + " ");
    }
}
Arrays as Parameters

• An entire array can be passed as a parameter to a method

• Like any other object, the reference to the array is passed, making the formal and actual parameters aliases of each other

• Therefore, changing an array element within the method changes the original

• An individual array element can be passed to a method as well, in which case the type of the formal parameter is the same as the element type
Arrays of Objects

• An array is an object and an array can hold objects as elements
• The array name is an object reference variable
• So this is another way to depict an array:
Arrays of Objects

• An array of objects really holds object references

• The following declaration reserves space to store 5 references to String objects

        String[] words = new String[5];

• It does not create the String objects themselves

• Initially an array of objects holds null references

• Each object stored in an array must be instantiated separately
Arrays of Objects

• After initial creation, an array holds null references:

• Each element is a reference to an object:
Arrays of Objects

• Keep in mind that `String` objects can be created using literals

• The following declaration creates an array object called `verbs` and fills it with four `String` objects created using string literals

```java
String[] verbs = {"play", "work", "eat", "sleep");
```

• The following example creates an array of `Grade` objects, each with a string representation and a numeric lower bound
public class GradeRange
{
    //----------------------------------------------------------------------------
    // Creates an array of Grade objects and prints them.
    //----------------------------------------------------------------------------
    public static void main(String[] args)
    {
        Grade[] grades =
        {
            new Grade("A", 95), new Grade("A-", 90),
            new Grade("B+", 87), new Grade("B", 85), new Grade("B-", 80),
            new Grade("C+", 77), new Grade("C", 75), new Grade("C-", 70),
            new Grade("D+", 67), new Grade("D", 65), new Grade("D-", 60),
            new Grade("F", 0)
        };

        for (Grade letterGrade : grades)
            System.out.println(letterGrade);
    }
}
public class Grade
{
    private String name;
    private int lowerBound;

    public Grade(String grade, int cutoff)
    {
        name = grade;
        lowerBound = cutoff;
    }

    public String toString()
    {
        return name + "\t" + lowerBound;
    }
}
public void setName(String grade) {
    name = grade;
}

public void setLowerBound(int cutoff) {
    lowerBound = cutoff;
}

public String getName() {
    return name;
}

public int getLowerBound() {
    return lowerBound;
}
Arrays of Objects

• Now let's look at an example that stores a collection of CD objects
public class Tunes
{
    // *-----------------------------------------------------------------
    //  Creates a CDCollection object and adds some CDs to it. Prints
    //  reports on the status of the collection.
    // *-----------------------------------------------------------------
    public static void main (String[] args)
    {
        CDCollection music = new CDCollection();

        music.addCD("Storm Front", "Billy Joel", 14.95, 10);
        music.addCD("Come On Over", "Shania Twain", 14.95, 16);
        music.addCD("Soundtrack", "Les Miserables", 17.95, 33);
        music.addCD("Graceland", "Paul Simon", 13.90, 11);

        System.out.println(music);

        music.addCD("Double Live", "Garth Brooks", 19.99, 26);
        music.addCD("Greatest Hits", "Jimmy Buffet", 15.95, 13);

        System.out.println(music);
    }
}
import java.text.NumberFormat;

public class CDCollection
{
    private CD[] collection;
    private int count;
    private double totalCost;

    // Constructor: Creates an initially empty collection.
    public CDCollection()
    {
        collection = new CD[100];
        count = 0;
        totalCost = 0.0;
    }
}
// Adds a CD to the collection, increasing the size of the
// collection if necessary.

public void addCD(String title, String artist, double cost, int tracks) {
    if (count == collection.length)
        increaseSize();

    collection[count] = new CD(title, artist, cost, tracks);
    totalCost += cost;
    count++;
}
public String toString()
{
    NumberFormat fmt = NumberFormat.getCurrencyInstance();

    String report = "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                   My CD Collection\n\n";
    report += "Number of CDs: " + count + "\n";
    report += "Total cost: " + fmt.format(totalCost) + "\n";
    report += "Average cost: " + fmt.format(totalCost/count);

    report += "$\n\nCD List:\n\n";

    for (int cd = 0; cd < count; cd++)
        report += collection[cd].toString() + "$\n";

    return report;
}
private void increaseSize() {
    CD[] temp = new CD[collection.length * 2];

    for (int cd = 0; cd < collection.length; cd++)
        temp[cd] = collection[cd];

    collection = temp;
}
public class CD
{
    private String title, artist;
    private double cost;
    private int tracks;

    //-----------------------------------------------
    // Creates a new CD with the specified information.
    //-----------------------------------------------
    public CD(String name, String singer, double price, int numTracks)
    {
        title = name;
        artist = singer;
        cost = price;
        tracks = numTracks;
    }
}
public String toString()
{
    NumberFormat fmt = NumberFormat.getCurrencyInstance();

    String description;

    description = fmt.format(cost) + "\t" + tracks + "\t";
    description += title + "\t" + artist;

    return description;
}
}
Command-Line Arguments

• The signature of the main method indicates that it takes an array of String objects as a parameter.
• These values come from command-line arguments that are provided when the interpreter is invoked.
• For example, the following invocation of the interpreter passes three String objects into main:

> java StateEval pennsylvania texas arizona

• These strings are stored at indexes 0-2 of the array parameter of the main method.
public class CommandLine
{
    // Prints all of the command line arguments provided by the user.
    public static void main(String[] args)
    {
        for (String arg : args)
            System.out.println(arg);
    }
}
Variable Length Parameter Lists

• Suppose we wanted to create a method that processed a different amount of data from one invocation to the next

• For example, let's define a method called `average` that returns the average of a set of integer parameters

```java
// one call to average three values
mean1 = average (42, 69, 37);

// another call to average seven values
mean2 = average (35, 43, 93, 23, 40, 21, 75);
```
Variable Length Parameter Lists

• We could define overloaded versions of the average method
  – Downside: we'd need a separate version of the method for each parameter count

• We could define the method to accept an array of integers
  – Downside: we'd have to create the array and store the integers prior to calling the method each time

• Instead, Java provides a convenient way to create variable length parameter lists
Variable Length Parameter Lists

• Using special syntax in the formal parameter list, we can define a method to accept any number of parameters of the same type

• For each call, the parameters are automatically put into an array for easy processing in the method
public double average(int ... list) {
    double result = 0.0;

    if (list.length != 0) {
        int sum = 0;
        for (int num : list)
            sum += num;
        result = (double) num / list.length;
    }

    return result;
}
Variable Length Parameter Lists

• The type of the parameter can be any primitive or object type

```java
public void printGrades(Grade ... grades)
{
    for (Grade letterGrade : grades)
        System.out.println (letterGrade);
}
```
Variable Length Parameter Lists

• A method that accepts a variable number of parameters can also accept other parameters.

• The following method accepts an `int`, a `String` object, and a variable number of `double` values into an array called `nums`.

```java
public void test(int count, String name, double ... nums)
{
    // whatever
}
```
Variable Length Parameter Lists

• The varying number of parameters must come last in the formal arguments

• A single method cannot accept two sets of varying parameters

• Constructors can also be set up to accept a variable number of parameters
public class VariableParameters
{
  //-----------------------------------------------------------------
  //  Creates two Family objects using a constructor that accepts
  //  a variable number of String objects as parameters.
  //-----------------------------------------------------------------
  public static void main(String[] args)
  {
    Family lewis = new Family("John", "Sharon", "Justin", "Kayla",
                             "Nathan", "Samantha");

    Family camden = new Family("Stephen", "Annie", "Matt", "Mary",
                               "Simon", "Lucy", "Ruthie", "Sam", "David");

    System.out.println(lewis);
    System.out.println();
    System.out.println(camden);
  }
}
public class Family
{
  private String[] members;

  // Constructor: Sets up this family by storing the (possibly
  // multiple) names that are passed in as parameters.
  public Family(String ... names)
  {
    members = names;
  }

  // Returns a string representation of this family.
  public String toString()
  {
    String result = "";

    for (String name : members)
    {
      result += name + "\n";
    }

    return result;
  }
}
Two-Dimensional Arrays

- A *one-dimensional array* stores a list of elements.
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns.
Two-Dimensional Arrays

• To be precise, in Java a two-dimensional array is an array of arrays

• A two-dimensional array is declared by specifying the size of each dimension separately

  ```java
  int[][] scores = new int[12][50];
  ```

• A array element is referenced using two index values

  ```java
  value = scores[3][6]
  ```

• The array stored in one row can be specified using one index
public class TwoDArray
{
    //-----------------------------------------------------------------
    // Creates a 2D array of integers, fills it with increasing
    // integer values, then prints them out.
    //-----------------------------------------------------------------
    public static void main(String[] args)
    {
        int[][] table = new int[5][10];

        // Load the table with values
        for (int row=0; row < table.length; row++)
            for (int col=0; col < table[row].length; col++)
                table[row][col] = row * 10 + col;

        // Print the table
        for (int row=0; row < table.length; row++)
        {
            for (int col=0; col < table[row].length; col++)
                System.out.print(table[row][col] + " \t");
            System.out.println();
        }
    }
}
## Two-Dimensional Arrays

<table>
<thead>
<tr>
<th>Expression</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>table</code></td>
<td><code>int[][]</code></td>
<td>2D array of integers, or array of integer arrays</td>
</tr>
<tr>
<td><code>table[5]</code></td>
<td><code>int[]</code></td>
<td>array of integers</td>
</tr>
<tr>
<td><code>table[5][12]</code></td>
<td><code>int</code></td>
<td>integer</td>
</tr>
</tbody>
</table>
import java.text.DecimalFormat;

public class SodaSurvey {

    public static void main (String[] args) {
        int[][] scores = { {3, 4, 5, 2, 1, 4, 3, 2, 4, 4},
                          {2, 4, 3, 4, 3, 3, 2, 1, 2, 2},
                          {3, 5, 4, 5, 5, 3, 2, 5, 5, 5},
                          {1, 1, 1, 3, 1, 2, 1, 3, 2, 4} };

        final int SODAS = scores.length;
        final int PEOPLE = scores[0].length;

        int[] sodaSum = new int[SODAS];
        int[] personSum = new int[PEOPLE];
for (int soda=0; soda < SODAS; soda++)
    for (int person=0; person < PEOPLE; person++)
    {
        sodaSum[soda] += scores[soda][person];
        personSum[person] += scores[soda][person];
    }

DecimalFormat fmt = new DecimalFormat("0.#");
System.out.println("Averages:
");

for (int soda=0; soda < SODAS; soda++)
    System.out.println("Soda #" + (soda+1) + ": " +
    fmt.format((float)sodaSum[soda]/PEOPLE));

System.out.println ();
for (int person=0; person < PEOPLE; person++)
    System.out.println("Person #" + (person+1) + ": " +
    fmt.format((float)personSum[person]/SODAS));
}
Multidimensional Arrays

• Any array with more than one dimension is a *multidimensional array*

• Each dimension subdivides the previous one into the specified number of elements

• Each dimension has its own `length` constant

• Because each dimension is an array of array references, the arrays within one dimension can be of different lengths
  
  – these are sometimes called *ragged arrays*
Multidimensional Arrays

• One way to visualize a four-dimensional array:

![Diagram of a four-dimensional array]

• Two-dimensional arrays are common, but beyond that usually an array has other objects involved.