Chapter 8 discusses the following main topics:

- Introduction to Arrays
- Processing Array Contents
- Passing Arrays as Arguments to Methods
- Some Useful Array Algorithms and Operations
- Returning Arrays from Methods
- String Arrays
- Arrays of Objects
Chapter Topics

Chapter 8 discusses the following main topics:

- The Sequential Search Algorithm
- Parallel Arrays
- Two-Dimensional Arrays
- Arrays with Three or More Dimensions
- The Selection Sort and the Binary Search
- Command-Line Arguments
- The ArrayList Class
Introduction to Arrays

• Primitive variables are designed to hold only one value at a time.
• Arrays allow us to create a collection of like values that are indexed.
• An array can store any type of data but only one type of data at a time.
• An array is a list of data elements.
Creating Arrays

• An array is an object so it needs an object reference.

```java
int[] numbers;
//declares a reference to an array that will hold integers.
```

• The next step creates the array and assigns its address to the `numbers` variable.

```java
numbers = new int[6];
//creates a new array that will hold 6 integers.
```

Array element values are initialized to 0.

Array indexes always start at 0.
Creating Arrays

• It is possible to declare an array reference and create it in the same statement.

```java
int[] numbers = new int[6];
```

• Arrays may be of any type, not just int.

```java
float[] temperatures = new float[100];
char[] letters = new char[41];
long[] units = new long[50];
double[] sizes = new double[1200];
```
Creating Arrays

- The array size must be a non-negative number.
- It may be a literal value or be derived from a constant or variable.

```java
final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```
- Once created, an array size is fixed and cannot be changed.
Accessing the Elements of an Array

- An array is accessed by:
  - the reference name
  - a subscript that identifies which element in the array to access.

```java
numbers[0] = 20; //pronounced "numbers sub zero"
```
Inputting and Outputting Array Elements

- Array elements can be treated as any other variable.
- They are simply accessed by the same name and a subscript.

Example: `ArrayDemo1.java`

- Array subscripts can be accessed using variables (such as for loop counters).

Example: `ArrayDemo2.java`
Bounds Checking

- Array indexes always start at zero and continue to (array length - 1).
  
  ```java
  int values = new int[10];
  ```

- This array would have indexes 0 through 9.

- Example: `InvalidSubscript.java`

- In for loops, it is typical to use `i`, `j`, and `k` as counting variables.
  
  - It might help to think of `i` as representing the word index.
Off-by-One Errors

• It is very easy to be off-by-one when accessing arrays.

```java
// This code has an off-by-one error.
int[] numbers = new int[100];
for (int i = 1; i <= 100; i++)
    numbers[i] = 99;
```

• Here, the equal sign allows the loop to continue on to index 100, where 99 is the last index in the array.

• This code would throw an `ArrayIndexOutOfBoundsException`.
Array Initialization

• When relatively few items need to be initialized, an initialization list can be used to initialize the array.

```java
int[] days = {31, 28, 31, 30, 31, 30, 31, 30, 31, 30, 31};
```

• The numbers in the list are stored in the array in order:

  • days[0] is assigned 31,
  • days[1] is assigned 28,
  • days[2] is assigned 31,
  • days[3] is assigned 30,
  • etc.

• Example: ArrayInitialization.java
Alternate Array Declaration

• Previously we showed arrays being declared:
  
  ```java
  int[] numbers;
  ```

  • However, the brackets can also go here:
    
    ```java
    int numbers[];
    ```

  • These are equivalent but the first style is typical.

• Multiple arrays can be declared on the same line.
  
  ```java
  int[] numbers, codes, scores;
  ```

• With the alternate notation each variable must have brackets.
  
  ```java
  int numbers[], codes[], scores;
  ```

• The scores variable in this instance is simply an int variable.
Processing Array Contents

• Processing data in an array is the same as any other variable.

  \[
grossPay = \text{hours}[3] \times \text{payRate};
\]

• Pre and post increment works the same:

  \[
  \text{int[]} \; \text{score} = \{7, 8, 9, 10, 11\};
  
  ++\text{score}[2]; \; // \; \text{Pre-increment \ operation}
  
  \text{score}[4]++; \; // \; \text{Post-increment \ operation}
  \]

• Example: PayArray.java
Processing Array Contents

• Array elements can be used in relational operations:

```java
if(cost[20] < cost[0])
{
    //statements
}
```

• They can be used as loop conditions:

```java
while(value[count] != 0)
{
    //statements
}
```
Array Length

- Arrays are objects and provide a public field named `length` that is a constant that can be tested.

```java
double[] temperatures = new double[25];
```
- The length of this array is 25.

- The length of an array can be obtained via its length constant.

```java
int size = temperatures.length;
```
- The variable `size` will contain 25.
The Enhanced for Loop

• Simplified array processing (read only)
• Always goes through all elements
• General:

> for(datatype elementVariable : array)  
>     statement;

Example:
int[] numbers = {3, 6, 9};
For(int val : numbers)  
{  
    System.out.println("The next value is" + val);
}
Array Size

- The length constant can be used in a loop to provide automatic bounding.

Index subscripts start at 0 and end at one less than the array length.

```java
for(int i = 0; i < temperatures.length; i++)
{
    System.out.println("Temperature "+i": "+temperatures[i]);
}
```
Array Size

- It is possible to get the size of an array from a user:

```java
String input;
int numTests;
int[] tests;
InputStreamReader reader =
    new InputStreamReader(System.in);
BufferedReader keyboard =
    new BufferedReader(reader);
System.out.print("How many numbers do you have? ");
input = keyboard.readLine();
umTests = Integer.parseInt(input);
tests = new int[numTests];
```

- Example: DisplayTestScores.java
Reassigning Array References

• An array reference can be assigned to another array of the same type.

    // Create an array referenced by the numbers variable.
    int[] numbers = new int[10];
    // Reassign numbers to a new array.
    numbers = new int[5];

• If the first (ten element) array no longer has a reference to it, it will be garbage collected.
The `numbers` variable holds the address of an `int` array.

```java
int[] numbers = new int[10];
```
Reassigning Array References

The `numbers` variable holds the address of an `int` array.

```java
numbers = new int[5];
```

This array gets marked for garbage collection.
Copying Arrays

This is not the way to copy an array.

```java
int[] array1 = { 2, 4, 6, 8, 10 };
int[] array2 = array1; // This does not copy array1.
```

Example: `SameArray.java`
Copying Arrays

• You cannot copy an array by merely assigning one reference variable to another.

• You need to copy the individual elements of one array to another.

```java
int[] firstArray = {5, 10, 15, 20, 25};
int[] secondArray = new int[5];
for (int i = 0; i < firstArray.length; i++)
    secondArray[i] = firstArray[i];
```

• This code copies each element of firstArray to the corresponding element of secondArray.
Passing Array Elements to a Method

• When a single element of an array is passed to a method it is handled like any other variable.

• Example: PassElements.java

• More often you will want to write methods to process array data by passing the entire array, not just one element at a time.
Passing Arrays as Arguments

- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.

Example: `PassArray.java`

```java
public static void showArray(int[] array) {
    for (int i = 0; i < array.length; i++)
        System.out.print(array[i] + " ");
}
```

```java
showArray(numbers);
```
Comparing Arrays

- The `==` operator only compares if the array references point to the same array object.

- To compare the contents of an array:

```java
int[] firstArray = { 2, 4, 6, 8, 10 };
int[] secondArray = { 2, 4, 6, 8, 10 }
boolean arraysEqual = true;
int i = 0;
if (firstArray.length != secondArray.length)
    arraysEqual = false;
while (arraysEqual && i < firstArray.length)
{
    if (firstArray[i] != secondArray[i])
        arraysEqual = false;
    i++;
}
if (arraysEqual)
    System.out.println("The arrays are equal.");
else
    System.out.println("The arrays are not equal.");
```
Useful Array Operations

• Finding the Highest Value

```java
int [] numbers = new int[50];
int highest = numbers[0];
for (int i = 1; i < numbers.length; i++)
{
    if (numbers[i] > highest)
        highest = numbers[i];
}
```

• Finding the Lowest Value

```java
int lowest = numbers[0];
for (int i = 1; i < numbers.length; i++)
{
    if (numbers[i] < lowest)
        lowest = numbers[i];
}
```
Useful Array Operations

• **Summing Array Elements:**
  ```java
ing int total = 0; // Initialize accumulator
  for (int i = 0; i < units.length; i++)
    total += units[i];
  ```

• **Averaging Array Elements:**
  ```java
double total = 0; // Initialize accumulator
  double average; // Will hold the average
  for (int i = 0; i < scores.length; i++)
    total += scores[i];
  average = total / scores.length;
  ```

• **Example:**  *SalesData.java, Sales.java*
Sorting an Array

- Java provides a class named Array that simplifies some array operations.
- The Array class has a static method named sort that will sort a numeric array in ascending order.
  \[\text{Array.sort(numbers);}\]
- To use the class, the import statement, \text{import java.util.array;} must be used.
Partially Filled Arrays

• Typically, if it is unknown how much data an array will be holding:
  • size the array to the largest expected number of elements.
  • use a counting variable to keep track of how much valid data is in the array.

```java
int[] array = new int[100];
int count = 0;

while (number != -1 && count <= 99) {
    count++;
    array[count - 1] = number;
}
```

input, number and keyboard were previously declared and keyboard references a BufferedReader object
Arrays and Files

• Saving the contents of an array to a file:

```java
int[] numbers = {10, 20, 30, 40, 50};

FileWriter fwriter = new FileWriter("Values.txt");
PrintWriter outputFile = new PrintWriter(fwriter);

for (int i = 0; i < numbers.length; i++)
    outputFile.println(numbers[i]);

outputFile.close();
```
Arrays and Files

- Reading the contents of a file into an array:

```java
int[] numbers = new int[5]; //assuming we know the size
String str;
int i = 0;
FileReader freader = new FileReader("Values.txt");
BufferedReader inputFile = new BufferedReader(freader);
str = inputFile.readLine();
while (str != null && i < numbers.length)
{
    numbers[i] = Integer.parseInt(str);
    i++;
    str = inputFile.readLine();
}
inputFile.close();
```
Returning an Array Reference

- A method can return a reference to an array.
- The return type of the method must be declared as an array of the right type.

```java
public static double[] getArray()
{
    double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };
    return array;
}
```
- The `getArray` method is a public static method that returns an array of doubles.
- Example: `ReturnArray.java`
String Arrays

- Arrays are not limited to primitive data.
- An array of String objects can be created:

```java
String[] names = { "Bill", "Susan", "Steven", "Jean" };  
```

The `names` variable holds the address to the array. A String array is an array of references to String objects.

Example:

`MonthDays.java`
String Arrays

If an initialization list is not provided, the new keyword must be used to create the array:

```java
String[] names = new String[4];
```

The `names` variable holds the address to the array.

- `names[0]`: null
- `names[1]`: null
- `names[2]`: null
- `names[3]`: null
String Arrays

- When an array is created in this manner, each element of the array must be initialized.

The names variable holds the address to the array.

```java
names[0] = "Bill";
names[1] = "Susan";
names[2] = "Steven";
names[3] = "Jean";
```
Calling String Methods On Array Elements

- String objects have several methods.
  - toUpperCase,
  - compareTo
  - equals
  - charAt

- Each element of a String array is a String object.

- Methods can be used by using the array name and index as before.

```
System.out.println(names[0].toUpperCase());
char letter = names[3].charAt(0);
```
The length Field &
The length Method

- Arrays have a final field named `length`.
- String objects have a method named `length`.
- To display the length of each string held in a String array:

  ```java
  for (int i = 0; i < names.length; i++)
    System.out.println(names[i].length());
  ```

- An array’s length is a field
  - You do not write a set of parentheses after its name.
- A String’s length is a method
  - You do write the parentheses after the name of the String class’s length method.
Arrays of Objects

• Since Strings are objects, we know that arrays can contain objects.

```java
BankAccount[] accounts = new BankAccount[5];
```

The `accounts` variable holds the address of an `BankAccount` array.

The array is an array of references to `BankAccount` objects.
Arrays of Objects

- Each element needs to be initialized.
  ```java
  for (int i = 0; i < accounts.length; i++)
      accounts[i] = new BankAccount();
  ```
- Example: `ObjectArray.java`

The `accounts` variable holds the address of an `BankAccount` array.
The Sequential Search Algorithm

• A search algorithm is a method of locating a specific item in a larger collection of data.

• The *sequential search algorithm* uses a loop to:
  • sequentially step through an array,
  • compare each element with the search value, and
  • stop when
    • the value is found or
    • the end of the array is encountered.

• Example: *SearchArray.java*
Parallel Arrays

- By using the same subscript, you can build relationships between data stored in two or more arrays.

```java
String[] names = new String[5];
String[] addresses = new String[5];
```

- The names array stores the names of five persons
- The addresses array stores the addresses of the same five persons.
- The data for one person is stored at the same index in each array.
Parallel Arrays

Relationship between names and addresses array elements.

- Parallel arrays are useful when storing data of unlike types.
- Example: `ParallelArrays.java`
Two-Dimensional Arrays

- A two-dimensional array is an array of arrays.
- It can be thought of as having rows and columns.

<table>
<thead>
<tr>
<th></th>
<th>column 0</th>
<th>column 1</th>
<th>column 2</th>
<th>column 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>row 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>row 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Two-Dimensional Arrays

• Declaring a two-dimensional array requires two sets of brackets and two size declarators
  • The first one is for the number of rows
  • The second one is for the number of columns.

\[
\text{double [][] scores = new double [3][4];}
\]

two dimensional array

• The two sets of brackets in the data type indicate that the scores variable will reference a two-dimensional array.

• Notice that each size declarator is enclosed in its own set of brackets.
Accessing Two-Dimensional Array Elements

- When processing the data in a two-dimensional array, each element has two subscripts:
  - one for its row and
  - another for its column.
Accessing Two-Dimensional Array Elements

The `scores` variable holds the address of a 2D array of `doubles`.

```
row 0
| scores[0][0] | scores[0][1] | scores[0][2] | scores[0][3] |
row 1
| scores[1][0] | scores[1][1] | scores[1][2] | scores[1][3] |
row 2
```

Address
Accessing Two-Dimensional Array Elements

The `scores` variable holds the address of a 2D array of `double`

<table>
<thead>
<tr>
<th>Address</th>
<th>row 0</th>
<th>row 1</th>
<th>row 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>column 0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>column 1</td>
<td>0</td>
<td>0</td>
<td>95</td>
</tr>
<tr>
<td>column 2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>column 3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Accessing one of the elements in a two-dimensional array requires the use of both subscripts.

`scores[2][1] = 95;`
Accessing Two-Dimensional Array Elements

- Programs that process two-dimensional arrays can do so with nested loops.

- To fill the scores array:

```java
for (int row = 0; row < 3; row++)
{
    for (int col = 0; col < 4; col++)
    {
        System.out.print("Enter a score: ");
        input = keyboard.readLine();
        scores[row][col] = Double.parseDouble(input);
    }
}
```
Accessing Two-Dimensional Array Elements

• To print out the scores array:

```java
for (int row = 0; row < 3; row++)
{
    for (int col = 0; col < 4; col++)
    {
        System.out.println(scores[row][col]);
    }
}
```

• Example: CorpSales.java
Initializing a Two-Dimensional Array

- Initializing a two-dimensional array requires enclosing each row’s initialization list in its own set of braces.

  ```java
  int[][] numbers = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
  ```

- Java automatically creates the array and fills its elements with the initialization values.

  - row 0    {1, 2, 3}
  - row 1    {4, 5, 6}
  - row 2    {7, 8, 9}

- Declares an array with three rows and three columns.
The *numbers* variable holds the address of a 2D array of `int` values.

```java
int[ ][ ] numbers = {{1, 2, 3},
                      {4, 5, 6},
                      {7, 8, 9}};
```

The diagram shows the initialization of the `numbers` array, with the corresponding addresses and values in a grid format: row `0` has elements `1`, `2`, `3`; row `1` has elements `4`, `5`, `6`; and row `2` has elements `7`, `8`, `9`. The addresses are shown for each element, illustrating how the array is allocated in memory.
The length Field

- Two-dimensional arrays are arrays of one-dimensional arrays.
- The length field of the array gives the number of rows in the array.
- Each row has a length constant tells how many columns is in that row.
- Each row can have a different number of columns.
The length Field

- To access the length fields of the array:

```java
int[][] numbers = {{ 1, 2, 3, 4 },
                    { 5, 6, 7 },
                    { 9, 10, 11, 12 }};

for (int row = 0; row < numbers.length; row++)
{
    for (int col = 0; col < numbers[row].length; col++)
        System.out.println(numbers[row][col]);
}
```

- Example: `Lengths.java`

The array can have variable length rows.
Summing The Elements of a Two-Dimensional Array

```java
int[][] numbers = { { 1, 2, 3, 4 },
                    {5, 6, 7, 8},
                    {9, 10, 11, 12} };  
int total;  
total = 0;  
for (int row = 0; row < numbers.length; row++)
    {
        for (int col = 0; col < numbers[row].length; col++)
            total += numbers[row][col];
    }

System.out.println("The total is " + total);
```
Summing The Rows of a Two-Dimensional Array

```java
int[][] numbers = {{ 1, 2, 3, 4},
                   {5, 6, 7, 8},
                   {9, 10, 11, 12}};

int total;

for (int row = 0; row < numbers.length; row++)
    {
        total = 0;
        for (int col = 0; col < numbers[row].length; col++)
            total += numbers[row][col];
        System.out.println("Total of row "+ row + " is "+ total);
    }
```
Summing The Columns of a Two-Dimensional Array

```java
int[][] numbers = {{1, 2, 3, 4},
                   {5, 6, 7, 8},
                   {9, 10, 11, 12}};

int total;

for (int col = 0; col < numbers[0].length; col++)
{
    total = 0;
    for (int row = 0; row < numbers.length; row++)
        total += numbers[row][col];
    System.out.println("Total of column "+ col + " is "+ total);
}
```
Passing and Returning Two-Dimensional Array References

• There is no difference between passing a single or two-dimensional array as an argument to a method.

• The method must accept a two-dimensional array as a parameter.

• Example: Pass2DArray.java
Ragged Arrays

• When the rows of a two-dimensional array are of different lengths, the array is known as a ragged array.

• You can create a ragged array by creating a two-dimensional array with a specific number of rows, but no columns.

```java
int [][] ragged = new int [4][];
```

• Then create the individual rows.

```java
ragged[0] = new int [3];
ragged[1] = new int [4];
ragged[2] = new int [5];
ragged[3] = new int [6];
```
More Than Two Dimensions

- Java does not limit the number of dimensions that an array may be.
- More than three dimensions is hard to visualize, but can be useful in some programming problems.
Selection Sort

- In a selection sort:
  - The smallest value in the array is located and moved to element 0.
  - Then the next smallest value is located and moved to element 1.
  - This process continues until all of the elements have been placed in their proper order.
- Example: `SelectionSortDemo.java`
Binary Search

A binary search:
- requires an array sorted in ascending order.
- starts with the element in the middle of the array.
- If that element is the desired value, the search is over.
- Otherwise, the value in the middle element is either greater or less than the desired value.
- If it is greater than the desired value, search in the first half of the array.
- Otherwise, search the last half of the array.
- Repeat as needed while adjusting start and end points of the search.

Example: **BinarySearchDemo.java**
Command-Line Arguments

- A Java program can receive arguments from the operating system command-line.
- The main method has a header that looks like this:
  ```java
  public static void main(String[] args)
  ```
- The main method receives a String array as a parameter.
- The array that is passed into the args parameter comes from the operating system command-line.
Command-Line Arguments

To run the example:

```java
java CommandLine How does this work?
```

- `args[0]` is assigned "How"
- `args[0]` is assigned "does"
- `args[0]` is assigned "this"
- `args[0]` is assigned "work?"

Examples: `CommandLine.java`

- `VarargsDemo2.java`

- It is not required that the name of main’s parameter array be `args`. 
The ArrayList Class

- Similar to Array, allows object storage
- Unlike Array, an ArrayList object:
  - Automatically expands when a new item is added
  - Automatically shrinks when items are removed
- Requires:
  - `import java.util.ArrayList;`
Creating and Using ArrayList

- Create ArrayList object with no-args constructor
  - ArrayList nameList = new ArrayList();
- To populate the ArrayList, use the add() method
  - nameList.add(“James”);
  - nameList.add(“Catherine”);
- To get the current size, call the size() method
  - nameList.size(); returns 2
- To access items in an ArrayList, use the get() method
  - nameList.get(1); where 1 is the index of the item
- Example: ArrayListDemo1.java
Using an ArrayList

- **ArrayList class toString() method**
  - Returns string representing all items in the ArrayList
  - `System.out.println(nameList);` yields
    - `[ James, Catherine ]`

- **The ArrayList class remove() method**
  - Removes designated item from the ArrayList
  - `nameList.remove(1);` removes second item

- **Example:** [ArrayListDemo3.java](#)
Using an ArrayList

- ArrayList class add() method with one argument adds new items to the end of the ArrayList.
- To insert items at a location of choice, use the add() method with two arguments.
  - nameList.add(1, "Mary"); inserts the new item at index 1.
  - Yielding [James, Mary, Catherine]
- To replace an existing item, use the set() method.
  - nameList.set(1, "Becky"); replaces "Mary" with "Becky".
- Example: ArrayListDemo4.java
Using an ArrayList

• Capacity and Capacity Increment
  • Default initial size of an ArrayList is 10 items
  • To designate initial size, use a parameterized constructor
    • ArrayList list = new ArrayList(100);
  • To designate initial size and size of increment:
    • ArrayList list = new ArrayList(100, 50);
Casting with ArrayList get() Method

- An ArrayList object is not typed
- To retrieve items from an ArrayList, you must cast the item to the appropriate type
  - ArrayList nameList = new ArrayList();
  - nameList.add("Mary"); inserts an item
  - String str = (String)nameList.get(0);
- Try get() without the cast to see the effect
- Example: ArrayListDemo6.java
Using ArrayList as a Generic Data Type

• We can create a type-safe ArrayList object by using generics

• For example an ArrayList object for Strings:
  • ArrayList<String> nameList = new ArrayList<String>();

• The get() no longer requires casts to work.

• Example: GenericArrayListDemo1.java

• Example: GenericArrayListDemo2.java