Arrays in Java

Selim Aksoy
Bilkent University
Department of Computer Engineering
saksoy@cs.bilkent.edu.tr

Arrays

- An array is an ordered list of values
- Each value has a numeric index
- The entire array has a single name
- An array of size N is indexed from zero to N-1

<table>
<thead>
<tr>
<th>scores</th>
</tr>
</thead>
<tbody>
<tr>
<td>79</td>
</tr>
<tr>
<td>87</td>
</tr>
<tr>
<td>94</td>
</tr>
<tr>
<td>82</td>
</tr>
<tr>
<td>67</td>
</tr>
<tr>
<td>98</td>
</tr>
<tr>
<td>87</td>
</tr>
<tr>
<td>81</td>
</tr>
<tr>
<td>74</td>
</tr>
<tr>
<td>91</td>
</tr>
</tbody>
</table>

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

\[
\text{scores[2]} = 89;
\]

\[
\text{scores[first]} = \text{scores[first]} + 2;
\]

\[
\text{mean} = (\text{scores[0]} + \text{scores[1]})/2;
\]

\[
\text{System.out.println("Top= "+ \text{scores[5]});}\]

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, or an array of characters, or an array of \text{String} objects, etc.
- In Java, the array itself is an object
- Therefore the name of the array is an object reference variable, and the array itself must be instantiated

Declaring Arrays

- The \text{scores} array could be declared as follows:
  \[ \text{int[]} \text{scores} = \text{new int[10]}; \]
- The type of the variable \text{scores} is \text{int[]} (an array of integers)
- Note that the type of the array does not specify its size, but each object of that type has a specific size
- The reference variable \text{scores} is set to a new array object that can hold 10 integers
Example

```java
public class BasicArray {
    final int LIMIT = 15;
    final int 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
    for (int index = 0; index < LIMIT; index++)
        System.out.print (list[index] + "  ");
}
```

Declaring Arrays

- Some examples of array declarations:
  ```java
  float[] prices = new float[500];
  boolean[] flags;
  flags = new boolean[20];
  char[] codes = new char[1750];
  ```

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 bounds (0 to N-1)
- The Java interpreter gives an error if an array index is out of bounds
- This is called automatic bounds checking

Example

```java
public class ReverseOrder {
    public static void main (String[] args) {
        double[] numbers = new double[10];
        System.out.println ("Size of array: " + numbers.length);
        for (int index = 0; index < numbers.length; index++)
            System.out.print ("Enter number " + (index+1) + " : ");
        numbers[index] = Keyboard.readDouble();
    }
    System.out.println ("The numbers in reverse order:");
    for (int index = numbers.length-1; index >= 0; index--)
        System.out.print (numbers[index] + "  ");
}
```
Example

// ************************************************************
//  LetterCount.java  Author: Lewis/Loftus
//  Demonstrates the relationship between arrays and strings.
// ************************************************************
import cs1.Keyboard;
public class LetterCount {
    // Reads 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;
        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 = Keyboard.readString();
        // 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
        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("Non-alphabetic characters:" + other);
    }
}

Initializer Lists

- An initializer list can be used to instantiate
  and initialize 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'};
  ```

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
- Changing an array element within the method
  changes the original
- An array element can be passed to a method
  as well, and follows the parameter passing
  rules of that element's type

Sorting

- Sorting is the process of arranging a list of items
  in a particular order
- The sorting process is based on specific value(s)
  - sorting a list of test scores in ascending numeric order
  - sorting a list of people alphabetically by last name
- There are many algorithms for sorting a list of
  items and these algorithms vary in efficiency
- We will examine two specific algorithms:
  - Selection Sort
  - Insertion Sort
Selection Sort

- The approach of Selection Sort:
  - select a value and put it in its final place in the list
  - repeat for all other values

- In more detail:
  - find the smallest value in the list
  - switch it with the value in the first position
  - find the next smallest value in the list
  - switch it with the value in the second position
  - repeat until all values are in their proper places

Example

```java
//***************************************************************
//  SortGrades.java Author: Lewis/Loftus
//  Driver for testing a numeric selection sort.
//***************************************************************
public class SortGrades {
  //Creates an array of grades, sorts them, then prints them.
  public static void main (String[] args) {
    int[] grades = {89, 94, 69, 80, 97, 85, 73, 91, 77, 85, 93};
    Sorts.selectionSort (grades);
    for (int index = 0; index < grades.length; index++)
      System.out.print (grades[index] + "   ");
  }
}
```

Selection Sort

- An example:
  - Original: 3 9 6 1 2
  - Smallest is 1: 1 9 6 3 2
  - Smallest is 2: 1 2 6 3 9
  - Smallest is 3: 1 2 3 6 9
  - Smallest is 6: 1 2 3 6 9

Swapping

- Swapping is the process of exchanging two values
- Swapping requires three assignment statements:
  ```java
temp = first;
first = second;
second = temp;
```

Insertion Sort

- The approach of Insertion Sort:
  - pick any item and insert it into its proper place in a sorted sublist
  - repeat until all items have been inserted

- In more detail:
  - consider the first item to be a sorted sublist (of one item)
  - insert the second item into the sorted sublist, shifting the first item as needed to make room to insert the new addition
  - insert the third item into the sorted sublist (of two items), shifting items as necessary
  - repeat until all values are inserted into their proper positions
Insertion Sort

- An example:

  original:  3 9 6 1 2
  insert 9:  3 9 6 1 2
  insert 6:  3 6 9 1 2
  insert 1:  1 3 6 9 2
  insert 2:  1 2 3 6 9

Comparing Sorts

- Both Selection and Insertion sorts are similar in efficiency.
- They both have outer loops that scan all elements, and inner loops that compare the value of the outer loop with almost all values in the list.
- Approximately \( n^2 \) number of comparisons are made to sort a list of size \( n \).
- We therefore say that these sorts are of order \( n^2 \).
- Other sorts are more efficient: order \( n \log_2 n \).