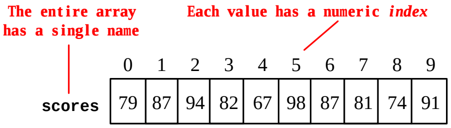
Arrays in Java

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

Arrays

- An *array* is an ordered list of values



The entire array has a single name

Each value has a numeric *index*

scores	0	1	2	3	4	5	6	7	8	9
	79	87	94	82	67	98	87	81	74	91

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
  `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

- For example, an array element can be assigned a value, printed, or used in a calculation:

```
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, or an array of characters, or an array of `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 `scores` array could be declared as follows:
  `int[] scores = new int[10];`
- The type of the variable `scores` is `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 `scores` is set to a new array object that can hold 10 integers

Example

```
//************************************************************
//  BasicArray.java       Author: Lewis/Loftus
//  Demonstrates basic array declaration and use.
//************************************************************
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;
      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:

```
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*

Bounds Checking

- For example, if the array `codes` can hold 100 values, it can be indexed using only the numbers 0 to 99
- If `count` has the value 100, then the following reference will cause an error:

  `System.out.println(codes[count]);`
- It's common to introduce *off-by-one errors* when using arrays

problem

```
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

Example

```
//*******************************************************************
//  ReverseOrder.java       Author: Lewis/Loftus
//  Demonstrates array index processing.
//*******************************************************************
import cs1.Keyboard;

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) {

      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();
```

Example

```
      //  Count the number of each letter occurence
      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:

```
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 only be used in the array declaration

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

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
```

Example

```
//************************************************************
//  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] + "   ");
   }
}
```

Example

```
//************************************************************
//  Sorts.java       Author: Lewis/Loftus
//  Demonstrates the selection sort and insertion sort algorithms,
//  as well as a generic object sort.
//************************************************************
public class Sorts {

   //Sorts the specified array of integers using the selection
   //sort algorithm.
   public static void selectionSort (int[] numbers) {

      int min, temp;

      for (int index = 0; index < numbers.length-1; index++) {

         min = index;
         for (int scan = index+1; scan < numbers.length; scan++) {
            if (numbers[scan] < numbers[min])
               min = scan;
         }

         // Swap the values
         temp = numbers[min];
         numbers[min] = numbers[index];
         numbers[index] = temp;
      }
   }
}
```

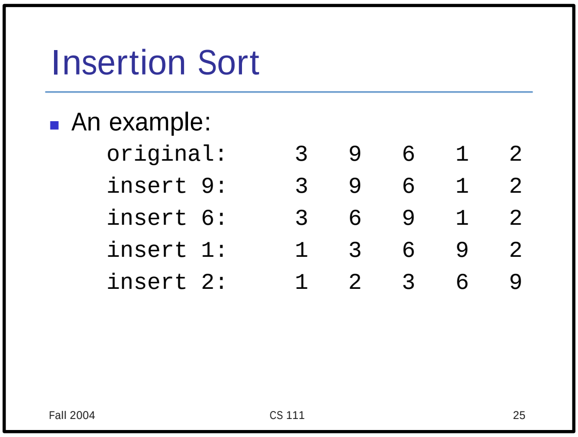
Swapping

- *Swapping* is the process of exchanging two values
- Swapping requires three assignment statements

```
temp = first;
first = second;
second = temp;
```

Insertion Sort

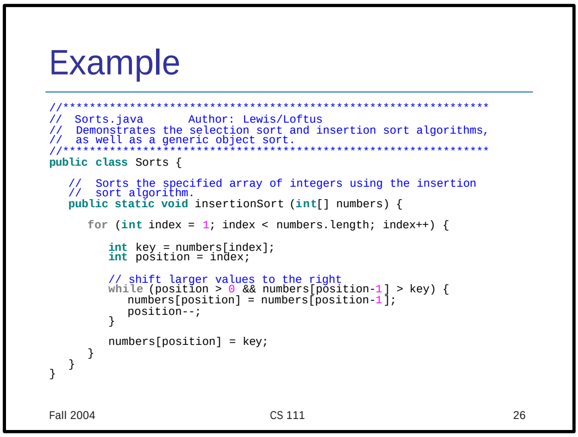
- The approach of Insertion Sort:
  - pick any item and insert it into its proper place in a sorted sublist
  - repeat until all items have been inserted
- In more detail:
  - consider the first item to be a sorted sublist (of one item)
  - insert the second item into the sorted sublist, shifting the 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

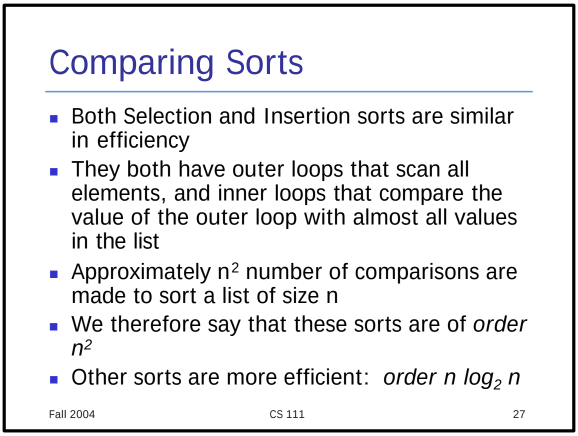
- An example:

```
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
```



Example

```
//****************************************************************
//  Sorts.java       Author: Lewis/Loftus
//  Demonstrates the selection sort and insertion sort algorithms,
//  as well as a generic object sort.
//****************************************************************
public class Sorts {

   //  Sorts the specified array of integers using the insertion
   //  sort algorithm.
   public static void insertionSort (int[] numbers) {

      for (int index = 1; index < numbers.length; index++) {

         int key = numbers[index];
         int position = index;

         // shift larger values to the right
         while (position > 0 && numbers[position-1] > key) {
            numbers[position] = numbers[position-1];
            position--;
         }

         numbers[position] = key;
      }
   }
}
```



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 $n \log_2 n$*