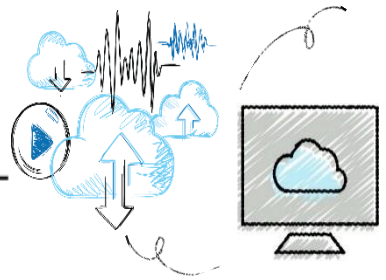
Java Programming, 9e

Chapter 9

Advanced Array Concepts



Objectives

- Sort array elements using the bubble sort algorithm
- Sort array elements using the insertion sort algorithm
- Use two-dimensional and other multidimensional arrays
- Use the `Arrays` class
- Use the `ArrayList` class
- Create enumerations

Sorting Array Elements Using the Bubble Sort Algorithm (1 of 2)

- **Sorting**
  - The process of arranging a series of objects in some logical order
- **Ascending order**
  - Begin with the object that has the lowest value
- **Descending order**
  - Begin with the object that has the largest value

Sorting Array Elements Using the Bubble Sort Algorithm (2 of 2)

- Simplest possible sort
  - Involves two values that are out of order
  - Swap two values (assume `valA = 16` and `valB = 2`)

```
temp = valA; // 16 goes to temp
valA = valB; // 2 goes to valA
valB = temp; // 16 goes to valB
```

Using the Bubble Sort Algorithm (1 of 3)

- **Bubble sort**
  - You continue to compare pairs of items, swapping them if they are out of order
  - The smallest items “bubble” to the top of the list, eventually creating a sorted list

Using the Bubble Sort Algorithm (2 of 3)

```
for(a = 0; a < someNums.length - 1; ++a)
   for(b = 0; b < someNums.length - 1; ++b)
      if(someNums[b] > someNums[b + 1])
      {
         temp = someNums[b];
         someNums[b] = someNums[b + 1];
         someNums[b + 1] = temp;
      }
```

Figure 9-1 Ascending bubble sort of the someNums array elements

Using the Bubble Sort Algorithm (3 of 3)

```
int comparisonsToMake = someNums.length - 1;
for(a = 0; a < someNums.length - 1; ++a)
{
   for(b = 0; b < comparisonsToMake; ++b)
   {
      if(someNums[b] > someNums[b + 1])
      {
         temp = someNums[b];
         someNums[b] = someNums[b + 1];
         someNums[b + 1] = temp;
      }
   }
   --comparisonsToMake;
}
```

The comparisonsToMake variable is decreased after each pass through the inner loop.

Figure 9-2 More efficient ascending bubble sort of the someNums array elements

Sorting Arrays of Objects

- You can sort arrays of objects in much the same way that you sort arrays of primitive types
  - Major difference
    - Make the comparison that determines whether to swap two array elements
    - Sort based on a particular object field

Sorting Array Elements Using the Insertion Sort Algorithm (1 of 2)

- **Insertion sort**
  - A sorting algorithm that enables you to look at each list element one at a time
  - Move items down if the tested element should be inserted prior to other elements
- Similar to the technique that sorts a group of objects manually

Sorting Array Elements Using the Insertion Sort Algorithm (2 of 2)

```
int[] someNums = {90, 85, 65, 95, 75};
a = 1;
while(a < someNums.length)
{
    temp = someNums[a];
    b = a - 1;
    while(b >= 0 && someNums[b] > temp)
    {
        someNums[b + 1] = someNums[b];
        --b;
    }
    someNums[b + 1] = temp;
    ++a;
}
```

Figure 9-6 The insertion sort

Using Two-Dimensional and Other Multidimensional Arrays (1 of 3)

- **One-dimensional** or **single-dimensional array**
  - An array that you can picture as a column of values
  - Elements are accessed using a single subscript
- **Two-dimensional arrays**
  - Have two or more columns of values
  - Have rows and columns
  - Use two subscripts
  - Are often called a **matrix** or **table**

```
int[][] someNumbers = new int[3][4];
```

Using Two-Dimensional and Other Multidimensional Arrays (2 of 3)

someNumbers[0][0]	someNumbers[0][1]	someNumbers[0][2]	someNumbers[0][3]
someNumbers[1][0]	someNumbers[1][1]	someNumbers[1][2]	someNumbers[1][3]
someNumbers[2][0]	someNumbers[2][1]	someNumbers[2][2]	someNumbers[2][3]

Figure 9-9 View of a two-dimensional array in memory

Using Two-Dimensional and Other Multidimensional Arrays (3 of 3)

```
int[][] rents = { {400, 450, 510},
                  {500, 560, 630},
                  {625, 676, 740},
                  {1000, 1250, 1600} };
```

Passing a Two-Dimensional Array to a Method

- Pass the array name just as you do with a one-dimensional array

```
public static void displayScores(int[][]scoresArray)
```

Using the `length` Field with a Two-Dimensional Array

- The `length` field holds the number of rows in the array

  `rents.length`

- Each row has a `length` field that holds the number of columns in the row

  `rents[1].length`

Understanding Jagged Arrays

- **Jagged array** (**ragged array**)
  - A two-dimensional array with rows of different lengths
- To create a jagged array:
  - Define the number of rows for a two-dimensional array
  - Do not define the number of columns in the rows
  - Then declare the individual rows

Using Other Multidimensional Arrays

- **Multidimensional arrays**
  - Arrays with more than one dimension
- Create arrays of any size
  - Keep track of the order of variables needed as subscripts
  - Do not exhaust your computer's memory

Using the `Arrays` Class (1 of 4)

- **`Arrays` class**
  - Contains many useful methods for manipulating arrays
  - `static` methods
    - Use them with the class name without instantiating an `Arrays` object
  - `binarySearch()` method
    - A convenient way to search through sorted lists of values of various data types
    - The list must be in order

Using the `Arrays` Class (2 of 4)

Table 9-2 Useful methods of the Arrays class	
Method	**Purpose**
static int binarySearch(type[] a, type key)	Searches the specified array for the specified key value using the binary search algorithm
static boolean equals(type[] a, type[] a2)	Returns true if the two specified arrays of the same type are equal to one another
static void fill(type[] a, type val)	Assigns the specified value to each element of the specified array
static void sort(type[] a)	Sorts the specified array into ascending order
static void sort(type[] a, int fromIndex, int toIndex)	Sorts the specified range of the array into ascending order
static void parallelSort(type[] a)	Sorts the specified array into ascending order
static void parallelSort(type[] a, int fromIndex, int toIndex)	Sorts the specified range of the array into ascending order

Using the Arrays Class (3 of 4)

```
import java.util.*;
public class ArrayDemo
{
   public static void main(String[] args)
   {
      int[] myScores = new int [5];
      display("Original array:                ", myScores);
      Arrays.fill(myScores, 8);
      display("After filling with 8s:         ", myScores);
      myScores[2] = 6;
      myScores[4] = 3;
      display("After changing two values:     ", myScores);
      Arrays.sort(myScores);
      display("After sorting:                 ", myScores);
   }
   public static void display(String message, int array[])
   {
      int sz = array.length;
      System.out.print(message);
      for(int x = 0; x < sz; ++x)
         System.out.print(array[x] + " ");
      System.out.println();
   }
}
```

Fills array with 8s

Sorts array values

Figure 9-15 The ArraysDemo application

Using the Arrays Class (4 of 4)

```
import java.util.*;
import javax.swing.*;
public class VerifyCode
{
   public static void main(String[] args)
   {
      char[] codes = {'B', 'E', 'K', 'M', 'P', 'T'};
      String entry;
      char userCode;
      int position;
      entry = JOptionPane.showInputDialog(null,
         "Enter a product code");
      usersCode = entry.charAt(0);
      position = Arrays.binarySearch(codes, userCode);
      if(position >= 0)
        JOptionPane.showMessageDialog(null, "Position of " +
          usersCode + " is " + position);
      else
         JOptionPane.showMessageDialog(null, usersCode +
           " is an invalid code");
   }
}
```

Figure 9-17 The VerifyCode application

Using the `ArrayList` Class (1 of 3)

- The **`ArrayList` class** provides some advantages over the `Arrays` class
  - **Dynamically resizable**
  - Can add an item at any point in an `ArrayList` container
  - Can remove an item at any point in an `ArrayList` container
- **Capacity**
  - The number of items an `ArrayList` can hold without having to increase its size

Using the `ArrayList` Class (2 of 3)

Table 9-3 Useful methods of the ArrayList class	
Method	**Purpose**
public void add(Object) public void add(int, Object)	Adds an item to an ArrayList; the default version adds an item at the next available location; an overloaded version allows you to specify a position at which to add the item
public void remove(int)	Removes an item from an ArrayList at a specified location
public void set(int, Object)	Alters an item at a specified ArrayList location
Object get(int)	Retrieves an item from a specified location in an ArrayList
Public int size()	Returns the current ArrayList size

Using the `ArrayList` Class (3 of 3)

```
import java.util.ArrayList;
public class ArrayListDemo
{
   public static void main(String[] args)
   {
      ArrayList<String> names = new ArrayList<String>();
      names.add("Abigail");
      display(names);
      names.add("Brian");
      display(names);
      names.add("Zachary");
      display(names);
      names.add(2, "Christy");
      display(names);
      names.remove(1);
      display(names);
      names.set(0, "Annette");
      display(names);
   }
   public static void display(ArrayList<String> names)
   {
      System.out.println("\nThe size of the list is " + names.size());
      for(int x = 0; x < names.size(); ++x)
         System.out.println("position " + x + " Name: " +
            names.get(x));
   }
}
```

Figure 9-20 The `ArrayListDemo` program

Creating Enumerations (1 of 7)

- **Enumerated data type**
  - A programmer-created data type with a fixed set of values
- To create an enumerated data type, use:
  - The keyword `enum`
  - An identifier for the type
  - A pair of curly braces that contain a list of the **`enum` constants**

```
enum Month {JAN, FEB, MAR, APR, MAY, JUN,
      JUL, AUG, SEP, OCT, NOV, DEC};
```

Creating Enumerations (2 of 7)

Table 9-4 Some useful nonstatic enum methods		
Method	**Description**	**Example if mon = Month.MAY**
toString()	Returns the name of the calling constant object	mon.toString() is "MAY"
ordinal()	Returns an integer that represents the constant's position in the list of constants; as with arrays, the first position is 0	mon.ordinal() is 4
equals()	Returns true if its argument is equal to the calling object's value	mon.equals(Month.MAY) is true mon.equals(Month.NOV) is false
compareTo()	Returns a negative integer if the calling object's ordinal value is less than that of the argument, 0 if they are the same, and a positive integer if the calling object's ordinal value is greater than that of the argument	mon.compareTo(Month.JUL) is negative mon.compareTo(Month.FEB) is positive mon.compareTo(Month.MAY) is 0

Creating Enumerations (3 of 7)

Table 9-5 Some static enum methods		
Method	**Description**	**Example with Month Enumeration**
valueOf()	Accepts a string parameter and returns an enumeration constant	Month.valueOf("DEC") returns the DEC enum constant
values()	Returns an array of the enumerated constants	Month.values() returns an array with 12 elements that contain the enum constants

Creating Enumerations (4 of 7)

```
import java.util.Scanner;
public class EnumDemo
{
   enum Month {JAN, FEB, MAR, APR, MAY, JUN,
      JUL, AUG, SEP, OCT, NOV, DEC};
   public static void main(String[] args)
   {
      Month birthMonth;
      String userEntry;
      int position;
      int comparison;
      Scanner input = new Scanner(System.in);
      System.out.println("The months are:");
      for(Month mon : Month.values())
         System.out.print(mon + " ");
      System.out.print("\n\nEnter the first three letter of " +
         "your birth month >> ");
      userEntry = input.nextline().toUpperCase();
      birthMonth = Month.valueOf(userEntry);
      System.out.println("You entered " + birthMonth);
      position = birthMonth.ordinal();
      System.out.println(birthMonth + " is in position " + position);
      System.out.println("So its month number is " + (position + 1));
      comparison = birthMonth.compareTo(Month.JUN);
      if(comparison < 0)
        System.out.println(birthMonth +
           " is earlier in the year than " + Month.JUN);
      else
         if(comparison > 0)
            System.out.println(birthMonth +
               " is later in the year than " + Month.JUN);
         else
            System.out.println(birthMonth + " is " + Month.JUN);
   }
}
```

A Month type variable is declared.

Enhanced for loop displays each value.

valueOf() converts string to an enumeration value.

ordinal() gets position of string.

compareTo() compares entered string to the JUN constant.

Figure 9-24 The EnumDemo class

Creating Enumerations (5 of 7)

- You can declare an enumerated type in its own file
  - Filename matches the type name and has a .java extension
- You can use comparison operators with enumeration constants
- You can use enumerations to control a switch structure

Creating Enumerations (6 of 7)

```
import java.util.Scanner;
public class EnumDemo2
{
   enum Property {SINGLE_FAMILY, MULTIPLE_FAMILY,
      CONDOMINIUM, LAND, BUSINESS};
   public static void main(String[] args)
   {
      Property propForSale = Property.MULTIPLE_FAMILY;
      switch(propForSale)
      {
         case SINGLE_FAMILY;
         case MULTIPLE_FAMILY;
            System.out.println("Listing fee is 5%");
            break;
         case CONDOMINIUM;
            System.out.println("Listing fee is 6%");
            break;
         case LAND;
         case BUSINESS:
            System.out.println
               ("We do not handle this type of property");
      }
   }
}
```

Figure 9-26 The EnumDemo2 class

Creating Enumerations (7 of 7)

- Advantages of creating an enumeration type:
  - Only allowed values can be assigned
  - Using `enum`s makes the values type-safe
  - Provides a form of self-documentation
  - You can also add methods and other fields to an `enum` type
- **Type-safe**
  - Describes a data type for which only appropriate behaviors are allowed

Don’t Do It

- Don’t forget that the first subscript used with a two-dimensional array represents the row, and that the second subscript represents the column
- Don’t try to store primitive data types in an `ArrayList` structure
- Don’t think `enum` constants are strings; they are not enclosed in quotes

Summary

- Sorting
  - The process of arranging a series of objects in some logical order
  - Bubble sort and Insertion sort
- Two-dimensional arrays
  - Both rows and columns
- `Arrays` class
- `ArrayList` class
- A programmer-created data type with a fixed set of values is an enumerated data type