CSI1102: Introduction to Software Design

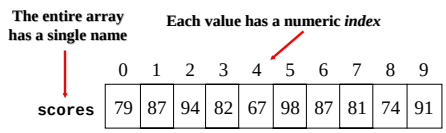
Chapter 6: Arrays

Learning objectives: Arrays (Chapter 6)

Understand how to do the following:

- Declaring and using Arrays
- Passing arrays and array elements as parameters
- Declaring and using Arrays of objects
- Sorting elements in an array: Selection and Insertion Sort
- Multidimensional arrays
- The `ArrayList` class
- Polygons and polylines, button components

Arrays: Ordered list of values



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

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

More about 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 (which is the 3rd value in the array)

- In Java, the array itself is an object
  - Therefore the **name** of the array is a **object reference variable**, and the **array itself must be instantiated**

Examples of Using Arrays

```
scores[2] = 89;

int first = 5;
scores[first] = scores[first] + 2;

mean = (scores[0] + scores[9])/2;

System.out.println ("Top = " + scores[5]);
```


More about Arrays

- An array stores multiple values **of the same type**
- That type can be **primitive types** or **object references**
- Therefore, we can create an **array of integers**, or an **array of characters**, or an **array of String objects**, etc.
- The `scores` array could be declared as follows

```
int[] scores = new int[10];
```

Declaring Arrays: More examples

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

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

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


Bounds Checking: Error 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 throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called *automatic bounds checking*

Bounds Checking: An example

- 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 exception to be thrown:

```
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: Using `length`

- Each array object has a public constant called `length` that stores the size of the array
- It is referenced using the array name (just like any other object):

```
scores.length
```

- Note that `length` holds the **number of elements**, not the largest index,
  - i.e. the value of **length** is (**largest index + 1**)

Bound checking: Extract from ReverseOrder.java (p.325)

```
{  double[] numbers = new double[10];

   System.out.println("Size :" + numbers.length);

   for (int index = 0; index < numbers.length; index++)
   {
      System.out.println("Enter number " + (index + 1) +  " : ");
      numbers[index] = Keyboard.readDouble());
   }
   // print in reverse order
   for (int index = numbers.length-1; index >= 0; index--)

   {
      System.out.println(numbers[index] +  "  ");
   }
}
```


Alternate Array Syntax: No need to do!

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following declarations are equivalent:

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

- **The first format generally is more readable!!!**

Initializer Lists: Setting up the values

- 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 only in the declaration of an array

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


Initializer lists: Extract from Primes.java (pp.330)

```
…
int[] primeNums = {2, 3, 5, 7, 11, 13, 17, 19};

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

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

for (int scan = 0; scan < primeNums.length; scan++)
  System.out.println(primeNums[scan] + " ");
  …
```


Using 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

Using Arrays of Objects

- The following declaration reserves space to store 25 references to `String` objects

```
String[] words = new String[25];
```

- It **does NOT create** the `String` objects themselves
- **Each object stored in an array must be instantiated separately**
- See `GradeRange.java` (page 332)

Arrays of Objects: Extract from GradeRange.java (pp.332)

```
…
int[] cutoff = {95, 90, 87, 83, 80};
String[] grades = ('A+', 'A', 'A-', 'B+', 'B');

for (int level = 0; level < cutoff.length; level++)
  System.out.println(grades[level] + " " + cutoff[level]);
…
```

Output

A+	95
A	90
A-	87
B+	83
B	80

Command-Line Arguments: About Main (at last!)

- 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 an array of three `String` objects into main:

```
> java DoIt pennsylvania texas california
```

- These strings are stored at indexes 0-2 of the parameter

About Main (String[] args): NameTag.java (p.334)

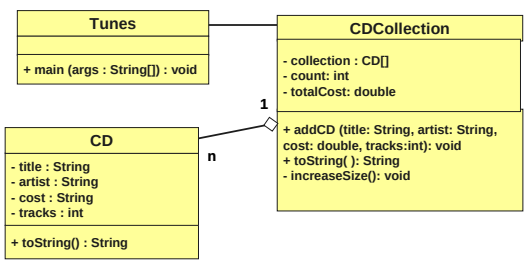
```
public class NameTag
{
   public static void main (String[] args)
   {
       System.out.println();
       System.out.println("          " + args[0]);
       System.out.println("My name is " + args[1]);
   }
}
```

```
> java NameTag Hello Sue
          Hello
My name is Sue
> java NameTag Hello James
          Hello
My name is James
```


Using Arrays of Objects

- Objects can have arrays as instance variables
- Therefore, many useful structures can be created simply with arrays and objects
- The software designer must determine carefully an organization of data and objects that makes sense for the situation
- See **CD Collection example**, p.335, 337-338

CD Selection Program: UML diagram



CD Collection example: Tunes.java (Driver program)

```
public class Tunes
   {
   public static void main (String[] args)
   {

   CDCollection music = new CDCollection();

   music.addCD("So far so good", "Bryan Adams", 14.96, 14);
   music.addCD("Enrique", "Enrique Iglesias", 15.96, 13);

   System.out.println(music);
   }
}
```


CD Collection example: CDCollection.java

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

   public CDCollection()        // the constructor
   {
       collection = new CD[100];
       count = 0;
       totalCost = 0.0;
   }
   //Adds a CD to the collection
   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++;
   }                                          Continued...
```


CD Collection example: CDCollection.java (cont.)

```
       // double the size of the collection
       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 String toString()
       {
           String report = " ";

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

           return report;
       }  }
```

CDCollection example: CD.java

```
public class CD
{
   private String title, artist;
   private double cost;
   private int tracks;

   // The constructor
   public CD (String name, String singer, double price, int numTracks)
   {
        title = name;
        artist = singer;
        cost = price;
        tracks = numTracks;
   }

   // the toString for printing
   public String toString()
   {
        String description;

        description = title + "\t" + artist;

        return description;
   }     }
```


Sorting of arrays

- Sorting is the process of arranging a list of items in a particular order
- There are many algorithms for sorting a list of items
- These algorithms vary in efficiency
- We will examine two specific algorithms:
  - Selection Sort
  - Insertion Sort

Selection Sort: The general idea

- The approach of Selection Sort:
  - select a value and put it in its final place into the sort 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: The Selection Sort method

- 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
```

- See `SortGrades.java` (page 342)
- See `Sorts.java` (page 343) -- the `selectionSort` method

Selection Sort: The selectionSort Method

```
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;
   }
}
```


Insertion Sort: The general idea

- 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

Sorting of Arrays: 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
```

- See Sorts.java (page 343) -- the `insertionSort` method

Insertion Sort: The insertionSort method

```
public static void insertionSort (int[] numbers)
{
   for (int index = 1; index < numbers.length; index++)
   {
        int key = numbers[index];
        int position = index;

        while (position > 0 && numbers[position – 1] > key)
        {
               numbers[position] = numbers[position – 1];
               position --;
        }
        numbers[position] = key;
   }
}
```


Comparing Sorts: Selection versus Insertion

- 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
- Therefore 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* $n \log_2 n$

Sorting Objects: An example

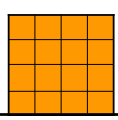
- The order of a collection of objects must be defined by the person defining the class
- Recall that a Java interface can be used as a type name and guarantees that a particular class implements particular methods
- We can use the `Comparable` interface and the `CompareTo` method to develop a **generic sort** for a set of objects
- See `SortPhoneList.java` and `Contact.java` (page 347-348)

More about Arrays: Two-Dimensional Arrays

- A *one-dimensional array* stores a **simple list of values**
- A *two-dimensional array* can be thought of as a **table** of values, with rows and columns
- Because each dimension is an array of array references, the arrays within one dimension can be of different lengths
  - Sometimes these are called *ragged arrays*
- A two-dimensional array element is referenced using two index values

```
int[][] scores = new int[10][10];
value = scores [3][6]
```

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

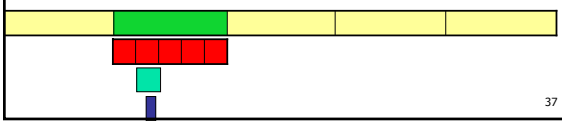


Two-Dimensional Arrays: TwoDArray.java

```
public class TwoDArray
{
// create a 2D array of integers and print them
public static void main (String[] args)
{
        int[][] table = new int[4][6];
        // load 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 values
        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();
        } … }}
```

Multidimensional Arrays

- An array can have **many dimensions**
- If it has more than one dimension, it is called a *multidimensional array*
- Each dimension subdivides the previous one into the specified number of elements
- Each array dimension has its own `length` constant
- This might be difficult for humans to visualize



The `ArrayList` Class: **Growing** and Shrinking

- The `ArrayList` class is part of the `java.util` package
- Like an array, it can store a **list of values** and reference them with an index
- Unlike an array, an `ArrayList` object ***grows*** and ***shrinks*** as needed
- Items can be inserted or removed with a single method invocation
- It stores references to the `Object` class

Some Methods of the `ArrayList` Class

```
ArrayList ()    // constructor: creates an empty list
boolean add (Object obj) // Add object to end of list
void add (int index, Object obj)
                // Add object at index
void clear();   // Remove all elements from the list
Object remove (int index);
                //Remove element at index
Object remove (int index);
          //Return element at index without removing it
Boolean isEmpty();   //Return true if the list is empty
```


`ArrayList` Efficiency

- The `ArrayList` class is implemented **using an array**.
- The array expands beyond its initial capacity to accommodate additional elements
- Methods manipulate the array so that indexes remain continuous as elements are added or removed

Using Arraylist: Beatles.java

```
import java.util.ArrayList;
public class Beatles                // stores and modify list of band members
{
      public static void main(String[] args)
      {
            ArrayList band = new ArrayList();

            band.add("Paul");
            band.add("Pete");
            band.add("John");
            band.add("George");
            System.out.println(band);

            int location = band.indexOf("Pete");
            band.remove(location);
            System.out.println(band);

            band.add("Ringo");
            System.out.println(band);
            System.out.println("Size of band: " + band.size());    }   } 41
```

Polygons and Polylines

- Arrays often are helpful in **graphics processing**
- Polygons and polylines are shapes that can be **defined by values stored in arrays**
- The `Polygon` class, defined in the `java.awt` package can be used to define and draw a polygon
- Two versions of the overloaded `drawPolygon` and `fillPolygon` methods take a single `Polygon` object as a parameter
- A `Polygon` object encapsulated the coordinates of the polygon
- See `Rocket.java` (page 360)

Other Button Components: Check boxes

- A *check box* is a button that can be toggled on or off
- A check box is represented by the `JCheckBox` class
- A change of state generates an *item event*

- See StyleOptions.java and StyleGUI.java (pp. 364-5)

Check boxes: Extract from StyleGUI.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
…
private JLabel saying;
private JCheckBox bold, italic;
private JPanel primary;
…
{
    saying = new JLabel("Say it with style!");
    bold = new JCheckBox("Bold");
    italic = new JCheckBox("Italic");
…
```

```
StyleListener listerner = new
    StyleListerner();

bold.addItemListener(listener);
italic.addItemListener(listener);

primary = new JPanel();
primary.add(saying);
primary.add(bold);
primary.add(italic);
primary.setBackground(Color.cyan);
primary.setPreferredSize(new
    Dimension(300, 100);
```

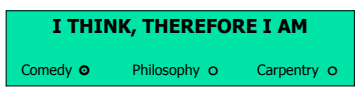

The Font Class

- A `Font` object is defined by the font name, the font style, and the font size
- The style of a font can be plain, bold, italic, or bold and italic together
- The `itemStateChanged` method of the listener responds when a check box changes state

```
final int FONT_SIZE = 32;

int style = font.PLAIN;

if (bold.isSelected())
    style = Font.BOLD;

saying.setFont (new Font("Tahoma", style,
FONT_SIZE);
```


Radio Buttons

- A set of *radio buttons* represents a set of mutually exclusive options
- When a radio button from a group is selected, the other button currently on in the group is toggled off
- A radio button generates an action event
- See `QuoteOptions.java` (page 368-369)



Summary: Chapter 6

- ARRAYS in Java
  - Array declaration and use
  - Passing arrays and array elements as parameters
  - Arrays of objects
  - Sorting elements in an array: selection and insertion sort
  - Multidimensional arrays
  - The `ArrayList` class
  - Polygons and polylines; more button components