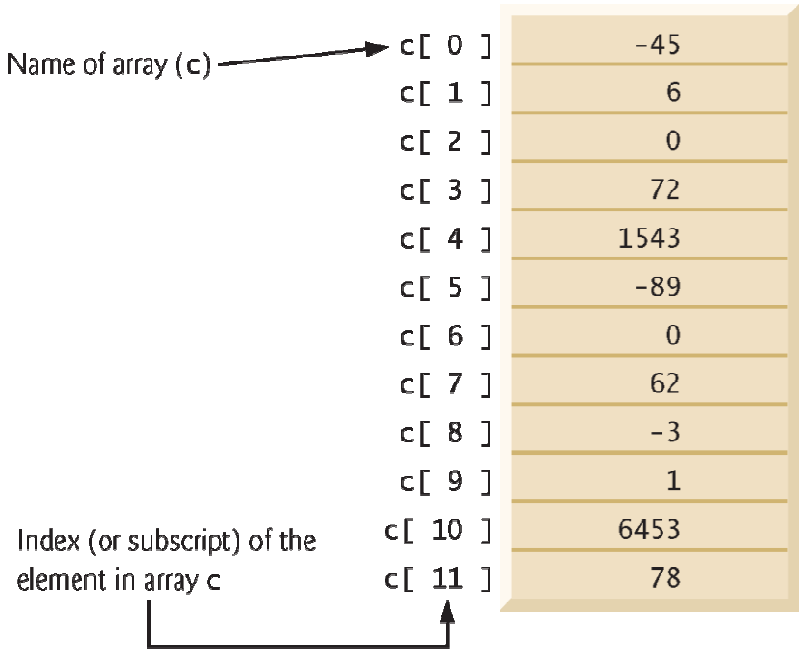
Array and ArrayList

Outline

- Arrays
- Creating arrays
- Using arrays
- Examples
- Multidimensional arrays
- Class Arrays
- Introduction to Collections and Class ArrayList

Arrays

- Array
  - Group of variables (called elements) containing values of the same type.
  - Arrays are objects so they are reference types.
  - Elements can be either primitive or reference types.
- Refer to a particular element in an array
  - Use the element's index.
  - Array-access expression—the name of the array followed by the index of the particular element in square brackets, [].
- The first element in every array has index zero.
- The highest index in an array is one less than the number of elements in the array.
- Array names follow the same conventions as other variable names.



A 12-element array.

UNIVERSITY OF WOLLONGONG

Arrays

- An index must be a nonnegative integer.
  - Can use an expression as an index.

```
int x=10;
c[x-2] = 5;
```

- An indexed array name is an array-access expression.
  - Can be used on the left side of an assignment to place a new value into an array element.
- Every array object knows its own length and stores it in a `length` instance variable.
  - `length` cannot be changed because it's a final variable.

UNIVERSITY OF WOLLONGONG

Declaring and Creating Arrays

- Array objects
  - Created with keyword new.
  - You specify the element type and the number of elements in an array-creation expression, which returns a reference that can be stored in an array variable.
- Declaration and array-creation expression for an array of 12 `int` elements

```
int[] c = new int[12];
```

- Can be performed in two steps as follows:

```
int[] c; // declare the array variable
c = new int[12]; // creates the array
```


UNIVERSITY OF WOLLONGONG

Declaring and Creating Arrays

- In a declaration, *square brackets* following a type indicate that a variable will refer to an array (i.e., store an array *reference*).
- When an array is created, each element of the array receives a default value
  - `Zero` for the numeric primitive-type elements, `false` for `boolean` elements and `null` for references.

Declaring and Creating Arrays

- Every element of a primitive-type array contains a value of the array's declared element type.
  - Every element of an `int` array is an `int` value.
- Every element of a reference-type array is a reference to an object of the array's declared element type.
  - Every element of a `String` array is a reference to a `String` object.

```
1  //            InitArray.java
2  // Initializing the elements of an array to default values of zero.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        // declare variable array and initialize it with an array object
9        int[] array = new int[10]; // create the array object
10
11       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
12
13       // output each array element's value
14       for (int counter = 0; counter < array.length; counter++)
15          System.out.printf("%5d%8d%n", counter, array[counter]);
16    }
17 } // end class InitArray
```

```
Index   Value
    0       0
    1       0
    2       0
    3       0
    4       0
    5       0
    6       0
    7       0
    8       0
    9       0
```


UNIVERSITY OF WOLLONGONG

Using an Array Initializer

- Array initializer
  - A comma-separated list of expressions (called an initializer list) enclosed in braces.
  - Used to create an array and initialize its elements.
  - Array length is determined by the number of elements in the initializer list.

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

    - Creates a five-element array with index values 0–4.
- Compiler counts the number of initializers in the list to determine the size of the array
  - Sets up the appropriate `new` operation "behind the scenes."

UNIVERSITY OF WOLLONGONG

```
1  //           InitArray.java
2  // Initializing the elements of an array with an array initializer.
3
4  public class InitArray
5  {
6     public static void main(String[] args)
7     {
8        // initializer list specifies the initial value for each element
9        int[] array = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
10
11       System.out.printf("%s%8s%n", "Index", "Value"); // column headings
12
13       // output each array element's value
14       for (int counter = 0; counter < array.length; counter++)
15          System.out.printf("%5d%8d%n", counter, array[counter]);
16
17 } // end class InitArray
```

```
Index   Value
    0      32
    1      27
    2      64
    3      18
    4      95
    5      14
    6      90
    7      70
    8      60
    9      37
```


UNIVERSITY OF WOLLONGONG

Using Bar Charts to Display Array Data Graphically

- Many programs present data to users in a graphical manner.
- Numeric values are often displayed as bars in a bar chart.
  - Longer bars represent proportionally larger numeric values.
- A simple way to display numeric data is with a bar chart that shows each numeric value as a bar of asterisks (*).
- Format specifier `%02d` indicates that an `int` value should be formatted as a field of two digits.
  - The 0 flag displays a leading 0 for values with fewer digits than the field width (2).

UNIVERSITY OF WOLLONGONG

```
1  //           BarChart.java
2  // Bar chart printing program.
3
4  public class BarChart
5  {
6     public static void main(String[] args)
7     {
8        int[] array = { 0, 0, 0, 0, 0, 0, 1, 2, 4, 2, 1 };
9
10       System.out.println("Grade distribution:");
11
12       // for each array element, output a bar of the chart
13       for (int counter = 0; counter < array.length; counter++)
14       {
15          // output bar label ("00-09: ", ..., "90-99: ", "100: ")
16          if (counter == 10)
17             System.out.printf("%5d: ", 100);
18          else
19             System.out.printf("%02d-%02d: ",
20                counter * 10, counter * 10 + 9);
21
22          // print bar of asterisks
23          for (int stars = 0; stars < array[counter]; stars++)
24             System.out.print("*");
25
26          System.out.println();
27       }
28    }
29 } // end class BarChart
```

```
Grade distribution:
00-09:
10-19:
20-29:
30-39:
40-49:
50-59:
60-69: *
70-79: **
80-89: ****
90-99: **
  100: *
```

Case Study: Card Shuffling and Dealing Simulation

- `Class Card` contains two `String` variables, i.e. face and suit
- Class `DeckOfCards` declares as an instance variable a `Card` array named `deck`.
- Deck's elements are `null` by default
  - Constructor fills the `deck` array with `Card` objects.
- Method `shuffle()` shuffles the Cards in the deck.
  - Loops through all 52 Cards (array indices 0 to 51).
  - Each Card swapped with a randomly chosen other card in the deck.
- Method `dealCard()` deals one `Card` in the array.
  - `currentCard` indicates the `index` of the next Card to be dealt
  - Returns `null` if there are no more cards to deal

```java
1  //            Card.java
2  // Card class represents a playing card.
3
4  public class Card
5  {
6     private final String face; // face of card ("Ace", "Deuce", ...)
7     private final String suit; // suit of card ("Hearts", "Diamonds", ...)
8
9     // two-argument constructor initializes card's face and suit
10    public Card(String cardFace, String cardSuit)
11    {
12       this.face = cardFace; // initialize face of card
13       this.suit = cardSuit; // initialize suit of card
14    }
15
16    // return String representation of Card
17    public String toString()
18    {
19       return face + " of " + suit;
20    }
21 } // end class Card
```

```
1  //              DeckOfCards.java
2  // DeckOfCards class represents a deck of playing cards.
3  import java.security.SecureRandom;
4
5  public class DeckOfCards
6  {
7     private Card[] deck; // array of Card objects
8     private int currentCard; // index of next Card to be dealt (0-51)
9     private static final int NUMBER_OF_CARDS = 52; // constant # of Cards
10    // random number generator
11    private static final SecureRandom randomNumbers = new SecureRandom();
12
13    // constructor fills deck of Cards
14    public DeckOfCards()
15    {
16       String[] faces = { "Ace", "Deuce", "Three", "Four", "Five", "Six",
17          "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King" };
18       String[] suits = { "Hearts", "Diamonds", "Clubs", "Spades" };
19
20       deck = new Card[NUMBER_OF_CARDS]; // create array of Card objects
21       currentCard = 0; // first Card dealt will be deck[0]
22
```

```
23          // populate deck with Card objects
24          for (int count = 0; count < deck.length; count++)
25             deck[count] =
26                new Card(faces[count % 13], suits[count / 13]);
27       }
28
29       // shuffle deck of Cards with one-pass algorithm
30       public void shuffle()
31       {
32          // next call to method dealCard should start at deck[0] again
33          currentCard = 0;
34
35          // for each Card, pick another random Card (0-51) and swap them
36          for (int first = 0; first < deck.length; first++)
37          {
38             // select a random number between 0 and 51
39             int second = randomNumbers.nextInt(NUMBER_OF_CARDS);
40
41             // swap current Card with randomly selected Card
42             Card temp = deck[first];
43             deck[first] = deck[second];
44             deck[second] = temp;
45          }
46       }
```

```
47
48       // deal one Card
49       public Card dealCard()
50       {
51          // determine whether Cards remain to be dealt
52          if (currentCard < deck.length)
53             return deck[currentCard++]; // return current Card in array
54          else
55             return null; // return null to indicate that all Cards were dealt
56       }
57    } // end class DeckOfCards
```

```
1  //            DeckOfCardsTest.java
2  // Card shuffling and dealing.
3
4  public class DeckOfCardsTest
5  {
6     // execute application
7     public static void main(String[] args)
8     {
9        DeckOfCards myDeckOfCards = new DeckOfCards();
10       myDeckOfCards.shuffle(); // place Cards in random order
11
12       // print all 52 Cards in the order in which they are dealt
13       for (int i = 1; i <= 52; i++)
14       {
15          // deal and display a Card
16          System.out.printf("%-19s", myDeckOfCards.dealCard());
17
18          if (i % 4 == 0) // output a newline after every fourth card
19             System.out.println();
20       }
21    }
22 } // end class DeckOfCardsTest
```

```
Six of Spades       Eight of Spades     Six of Clubs        Nine of Hearts
Queen of Hearts     Seven of Clubs      Nine of Spades      King of Hearts
Three of Diamonds   Deuce of Clubs      Ace of Hearts       Ten of Spades
Four of Spades      Ace of Clubs        Seven of Diamonds   Four of Hearts
Three of Clubs      Deuce of Hearts     Five of Spades      Jack of Diamonds
King of Clubs       Ten of Hearts       Three of Hearts     Six of Diamonds
Queen of Clubs      Eight of Diamonds   Deuce of Diamonds   Ten of Diamonds
Three of Spades     King of Diamonds    Nine of Clubs       Six of Hearts
Ace of Spades       Four of Diamonds    Seven of Hearts     Eight of Clubs
Deuce of Spades     Eight of Hearts     Five of Hearts      Queen of Spades
Jack of Hearts      Seven of Spades     Four of Clubs       Nine of Diamonds
Ace of Diamonds     Queen of Diamonds   Five of Clubs       King of Spades
Five of Diamonds    Ten of Clubs        Jack of Spades      Jack of Clubs
```

Enhanced `for` Statement

- Enhanced `for` statement
  - Iterates through the elements of an array without using a counter.
  - Avoids the possibility of "stepping outside" the array.
- Syntax:

```
for (parameter : arrayName)
   statement
```

  where *parameter* has a type and an identifier and *arrayName* is the array through which to iterate.
- Parameter type must be consistent with the array's element type.
- The enhanced for statement simplifies the code for iterating through an array.

```
1  //             EnhancedForTest.java
2  // Using the enhanced for statement to total integers in an array.
3
4  public class EnhancedForTest
5  {
6     public static void main(String[] args)
7     {
8        int[] array = { 87, 68, 94, 100, 83, 78, 85, 91, 76, 87 };
9        int total = 0;
10
11       // add each element's value to total
12       for (int number : array)
13          total += number;
14
15       System.out.printf("Total of array elements: %d%n", total);
16    }
17 } // end class EnhancedForTest
```

```
Total of array elements: 849
```

```
for (int counter=0; counter< array.length; counter++)
        total += array[counter];
```

Enhanced `for` Statement

- The enhanced `for` statement can be used *only* to obtain array elements
  - It *cannot* be used to *modify* elements.
  - To modify elements, use the traditional counter-controlled for statement.
- Can be used in place of the counter-controlled `for` statement if you don't need to access the index of the element.

Passing Arrays to Methods

- To pass an array argument to a method, specify the name of the array without any brackets.
  - Since every array object "knows" its own length, we need not pass the array length as an additional argument.
- To receive an array, the method's parameter list must specify an *array parameter*.
- When an argument to a method is an entire array or an individual array element of a reference type, the called method receives a copy of the reference.
- When an argument to a method is an individual array element of a primitive type, the called method receives a copy of the element's value.

```
1  //                PassArray.java
2  // Passing arrays and individual array elements to methods.
3
4  public class PassArray
5  {
6     // main creates array and calls modifyArray and modifyElement
7     public static void main(String[] args)
8     {
9        int[] array = { 1, 2, 3, 4, 5 };
10
11       System.out.printf(
12          "Effects of passing reference to entire array:%n" +
13          "The values of the original array are:%n");
14
15       // output original array elements
16       for (int value : array)
17          System.out.printf("   %d", value);
18
19       modifyArray(array); // pass array reference
20       System.out.printf("%n%nThe values of the modified array are:%n");
21
```

```
22          // output modified array elements
23          for (int value : array)
24             System.out.printf("   %d", value);
25
26          System.out.printf(
27             "%n%nEffects of passing array element value:%n" +
28             "array[3] before modifyElement: %d%n", array[3]);
29
30          modifyElement(array[3]); // attempt to modify array[3]
31          System.out.printf(
32             "array[3] after modifyElement: %d%n", array[3]);
33       }
34
35       // multiply each element of an array by 2
36       public static void modifyArray(int[] array2)
37       {
38          for (int counter = 0; counter < array2.length; counter++)
39             array2[counter] *= 2;
40       }
41
42       // multiply argument by 2
43       public static void modifyElement(int element)
44       {
45          element *= 2;
46          System.out.printf(
47             "Value of element in modifyElement: %d%n", element);
48       }
49    } // end class PassArray
```


UNIVERSITY OF WOLLONGONG

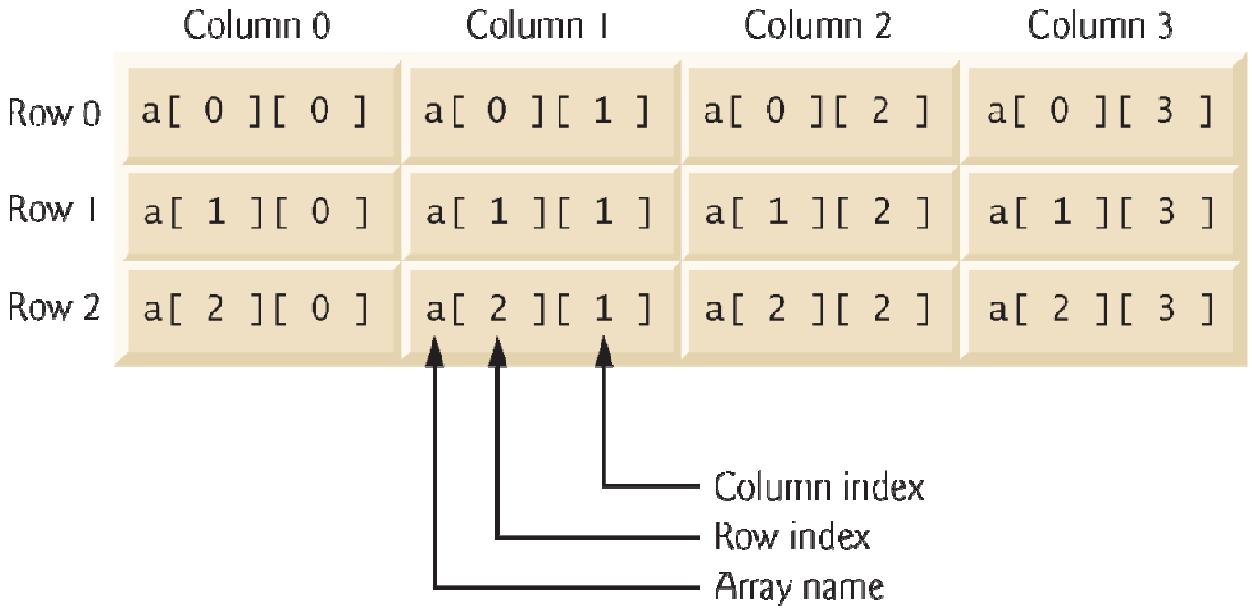
```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

Multidimensional Arrays

- Two-dimensional arrays are often used to represent tables of values with data arranged in *rows* and *columns*.
- Identify each table element with two indices.
  - By convention, the first identifies the element's row and the second its column.
- Multidimensional arrays can have more than two dimensions.
- Java does not support multidimensional arrays directly
  - Allows you to specify one-dimensional arrays whose elements are also one-dimensional arrays, thus achieving the same effect.
- In general, an array with *m* rows and *n* columns is called an *m*-by-*n* array.

  int[][] a = new int[2][3]

UNIVERSITY OF WOLLONGONG


Column 0
Column 1
Column 2
Column 3
Row 0
a[0][0]
a[0][1]
a[0][2]
a[0][3]
Row 1
a[1][0]
a[1][1]
a[1][2]
a[1][3]
Row 2
a[2][0]
a[2][1]
a[2][2]
a[2][3]
Column index
Row index
Array name

Multidimensional Arrays

- Multidimensional arrays can be initialized with array initializers in declarations.
- A two-dimensional array b with two rows and two columns could be declared and initialized with nested array initializers as follows:

```
int[][] b = {{1, 2}, {3, 4}};
```

  - The initial values are *grouped by row* in braces.
  - The number of nested array initializers (represented by sets of braces within the outer braces) determines the number of *rows*.
  - The number of initializer values in the nested array initializer for a row determines the number of *columns* in that row.
  - *Rows can have different lengths.*

UNIVERSITY OF WOLLONGONG

Multidimensional Arrays

- The lengths of the rows in a two-dimensional array are not required to be the same:

```
int[][] b = {{1, 2}, {3, 4, 5}};
```

  - Each element of b is a reference to a one-dimensional array of int variables.
  - The int array for row 0 (`b[0]`) is a one-dimensional array with two elements (1 and 2).
  - The int array for row 1 (`b[1]`) is a one-dimensional array with three elements (3, 4 and 5).

UNIVERSITY OF WOLLONGONG

Multidimensional Arrays

- A multidimensional array with the same number of columns in every row can be created with an array-creation expression.

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

  - 3 rows and 4 columns.
- The elements of a multidimensional array are initialized when the array object is created.
- A multidimensional array in which each row has a different number of columns can be created as follows:

```
int[][] b = new int[2][];   // create 2 rows
b[0] = new int[5]; // create 5 columns for row 0
b[1] = new int[3]; // create 3 columns for row 1
```

  - Creates a two-dimensional array with two rows.
  - Row 0 has five columns, and row 1 has three columns.

```
1  //             InitArray.java
2  // Initializing two-dimensional arrays.
3
4  public class InitArray
5  {
6     // create and output two-dimensional arrays
7     public static void main(String[] args)
8     {
9        int[][] array1 = {{1, 2, 3}, {4, 5, 6}};
10       int[][] array2 = {{1, 2}, {3}, {4, 5, 6}};
11
12       System.out.println("Values in array1 by row are");
13       outputArray(array1); // displays array1 by row
14
15       System.out.printf("%nValues in array2 by row are%n");
16       outputArray(array2); // displays array2 by row
17    }
18
19    // output rows and columns of a two-dimensional array
20    public static void outputArray(int[][] array)
21    {
22       // loop through array's rows
23       for (int row = 0; row < array.length; row++)
24       {
25          // loop through columns of current row
26          for (int column = 0; column < array[row].length; column++)
27             System.out.printf("%d  ", array[row][column]);
28
29          System.out.println();
30       }
31    }
32 } // end class InitArray
```

```
Values in array1 by row are
1  2  3
4  5  6

Values in array2 by row are
1  2
3
4  5  6
```


UNIVERSITY OF WOLLONGONG

Class `Arrays`

- `Arrays` class
  - Provides static methods for common array manipulations.
- Methods include
  - `sort` for sorting an array (ascending order by default)
  - `binarySearch` for searching a sorted array
  - `equals` for comparing arrays
  - `fill` for placing values into an array.
- Methods are overloaded for primitive-type arrays and for arrays of objects.
- `static arraycopy` method
  - Copies contents of one array into another.

UNIVERSITY OF WOLLONGONG

```
1  //            ArrayManipulations.java
2  // Arrays class methods and System.arraycopy.
3  import java.util.Arrays;
4
5  public class ArrayManipulations
6  {
7     public static void main(String[] args)
8     {
9        // sort doubleArray into ascending order
10       double[] doubleArray = { 8.4, 9.3, 0.2, 7.9, 3.4 };
11       Arrays.sort(doubleArray);
12       System.out.printf("%ndoubleArray: ");
13
14       for (double value : doubleArray)
15          System.out.printf("%.1f ", value);
16
17       // fill 10-element array with 7s
18       int[] filledIntArray = new int[10];
19       Arrays.fill(filledIntArray, 7);
20       displayArray(filledIntArray, "filledIntArray");
21
```

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
```

```
filledIntArray: 7 7 7 7 7 7 7 7 7 7
```

```
22          // copy array intArray into array intArrayCopy
23          int[] intArray = { 1, 2, 3, 4, 5, 6 };
24          int[] intArrayCopy = new int[intArray.length];
25          System.arraycopy(intArray, 0, intArrayCopy, 0, intArray.length);
26          displayArray(intArray, "intArray");
27          displayArray(intArrayCopy, "intArrayCopy");
28
29          // compare intArray and intArrayCopy for equality
30          boolean b = Arrays.equals(intArray, intArrayCopy);
31          System.out.printf("%n%nintArray %s intArrayCopy%n",
32             (b ? "==" : "!="));
33
34          // compare intArray and filledIntArray for equality
35          b = Arrays.equals(intArray, filledIntArray);
36          System.out.printf("intArray %s filledIntArray%n",
37             (b ? "==" : "!="));
38
39          // search intArray for the value 5
40          int location = Arrays.binarySearch(intArray, 5);
41
42          if (location >= 0)
43             System.out.printf(
44                "Found 5 at element %d in intArray%n", location);
45          else
46             System.out.println("5 not found in intArray");
```

```
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6
```

```
intArray == intArrayCopy
```

```
intArray != filledIntArray
```

```
Found 5 at element 4 in intArray
```

```
if (b)
   System.out.printf("intArray == filledIntArray %n");
else
   System.out.printf("intArray != filledIntArray %n");
```

```
47
48          // search intArray for the value 8763
49          location = Arrays.binarySearch(intArray, 8763);
50
51          if (location >= 0)
52             System.out.printf(
53                "Found 8763 at element %d in intArray%n", location);
54          else
55             System.out.println("8763 not found in intArray");
56       }
57
58       // output values in each array
59       public static void displayArray(int[] array, String description)
60       {
61          System.out.printf("%n%s: ", description);
62
63          for (int value : array)
64             System.out.printf("%d ", value);
65       }
66    } // end class ArrayManipulations
```

```
8763 not found in intArray
```

```
doubleArray: 0.2 3.4 7.9 8.4 9.3
filledIntArray: 7 7 7 7 7 7 7 7 7 7
intArray: 1 2 3 4 5 6
intArrayCopy: 1 2 3 4 5 6

intArray == intArrayCopy
intArray != filledIntArray
Found 5 at element 4 in intArray
8763 not found in intArray
```


UNIVERSITY OF WOLLONGONG

Introduction to Collections and Class `ArrayList`

- Java API provides several predefined data structures, called collections, used to store groups of related objects in memory.
  - Each provides efficient methods that organize, store and retrieve your data without requiring knowledge of how the data is being stored.
  - Reduce application-development time.
- Arrays do not automatically change their size at execution time to accommodate additional elements.
- `ArrayList<T>` (package `java.util`) can dynamically change its size to accommodate more elements.
  - T is a placeholder for the type of element stored in the collection.
- Classes with this kind of placeholder that can be used with any type are called generic classes.

Method	Description
`add`	Adds an element to the *end* of the `ArrayList`.
`clear`	Removes all the elements from the `ArrayList`.
`contains`	Returns `true` if the `ArrayList` contains the specified element; otherwise, returns `false`.
`get`	Returns the element at the specified index.
`indexOf`	Returns the index of the first occurrence of the specified element in the `ArrayList`.
`remove`	Overloaded. Removes the first occurrence of the specified value or the element at the specified index.
`size`	Returns the number of elements stored in the `ArrayList`.
`trimToSize`	Trims the capacity of the `ArrayList` to the current number of elements.

Introduction to Collections and Class `ArrayList`

- Method `add` adds elements to the `ArrayList`.
  - One-argument version appends its argument to the end of the `ArrayList`.
  - Two-argument version inserts a new element at the specified position.
  - Collection indices start at zero.
- Method `size` returns the number of elements in the ArrayList.
- Method `get` obtains the element at a specified index.
- Method `remove` deletes an element with a specific value.
  - An overloaded version of the method removes the element at the specified index.
- Method `contains` determines if an item is in the `ArrayList`.

```
1  //            ArrayListCollection.java
2  // Generic ArrayList<T> collection demonstration.
3  import java.util.ArrayList;
4
5  public class ArrayListCollection
6  {
7     public static void main(String[] args)
8     {
9        // create a new ArrayList of Strings with an initial capacity of 10
10       ArrayList<String> items = new ArrayList<String>();
11
12       items.add("red"); // append an item to the list
13       items.add(0, "yellow"); // insert "yellow" at index 0
14
15       // header
16       System.out.print(
17          "Display list contents with counter-controlled loop:");
18
19       // display the colors in the list
20       for (int i = 0; i < items.size(); i++)
21          System.out.printf(" %s", items.get(i));
22
```

```
Display list contents with counter-controlled loop: yellow red
```

```
23          // display colors using enhanced for in the display method
24          display(items,
25             "%nDisplay list contents with enhanced for statement:");
26                                  Display list contents with enhanced for statement: yellow red
27          items.add("green"); // add "green" to the end of the list
28          items.add("yellow"); // add "yellow" to the end of the list
29          display(items, "List with two new elements:");
30                                          List with two new elements: yellow red green yellow
31          items.remove("yellow"); // remove the first "yellow"
32          display(items, "Remove first instance of yellow:");
33                                       Remove first instance of yellow: red green yellow
34          items.remove(1); // remove item at index 1
35          display(items, "Remove second list element (green):");
36                                             Remove second list element (green): red yellow
37          // check if a value is in the List
38          System.out.printf("\"red\" is %sin the list%n",
39             items.contains("red") ? "": "not ");                    "red" is in the list
40
41          // display number of elements in the List
42          System.out.printf("Size: %s%n", items.size());
43       }                                                                    Size: 2
44
```

```
if (items.contains("red"))
   System.out.printf("red is in the list");
else
   System.out.printf("red is in the list");
```


UNIVERSITY OF WOLLONGONG

```
45      // display the ArrayList's elements on the console
46      public static void display(ArrayList<String> items, String header)
47      {
48         System.out.printf(header); // display header
49
50         // display each element in items
51         for (String item : items)
52            System.out.printf(" %s", item);
53
54         System.out.println();
55      }
56   } // end class ArrayListCollection
```

```
Display list contents with counter-controlled loop: yellow red
Display list contents with enhanced for statement: yellow red
List with two new elements: yellow red green yellow
Remove first instance of yellow: red green yellow
Remove second list element (green): red yellow
"red" is in the list
Size: 2
```

Suggested reading

Java: How to Program (Early Objects), 10th Edition

- Chapter 7: Arrays and ArrayLists
  - 7.1 – 7.4, 7.6 - 7.9, 7.11, 7.15, 7.16