



Chapter 7

Single-Dimensional Arrays

Opening Problem

- Read one hundred numbers, compute their average, and find out how many numbers are above the average.
- Java and most other high-level languages provide a data structure, the **array**, which stores *a fixed-size sequential collection of elements of the same type*.
- A single *array variable* can reference a large collection of data.
- This chapter introduces single-dimensional arrays. The next chapter will introduce two-dimensional and multidimensional arrays.

Objectives (1)

- To describe why arrays are necessary in programming (§7.1).
- To declare **array reference variables** and create arrays (§§7.2.1–7.2.2).
- To obtain array size using *arrayRefVar.length* and know default values in an array (§7.2.3).
- To access array elements using indexes (§7.2.4).
- To declare, create, and initialize an array using an array initializer (§7.2.5).
- To program common array operations (displaying arrays, summing all elements, finding the minimum and maximum elements, random shuffling, and shifting elements) (§7.2.6).
- To simplify programming using the **foreach loops** (§7.2.7).

3

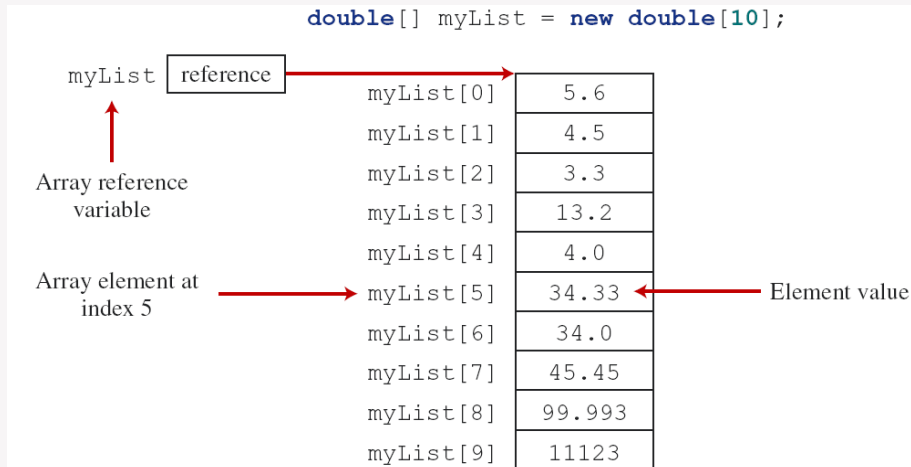
Objectives (2)

- To apply arrays in application development (AnalyzeNumbers, DeckOfCards) (§§7.3–7.4).
- To copy contents from one array to another (§7.5).
- To develop and invoke methods with **array arguments and return values** (§§7.6–7.8).
- To define a method with a **variable-length argument list** (§7.9).
- To search elements using the linear (§7.10.1) or binary (§7.10.2) search algorithm.
- To sort an array using the selection sort approach (§7.11).
- To use the methods in the *java.util.Arrays* class (§7.12).
- To pass arguments to the main method from the command line (§7.13).

4

7.2 Array Basics

- Array is a data structure that represents a collection of the same types of data.



5

Declaring Array Variables

- To use an array in a program, you must declare a variable to reference the array and specify the array's element type.

`datatype[] arrayRefVar;`

- Example: `double[] myList;`

`datatype arrayRefVar[];`

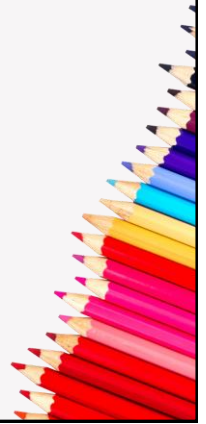
// This style is allowed, but not preferred

- Example: `double myList[];`

6

Creating Arrays

- The declaration of an array variable does not allocate any space in memory for the array.
- It creates only a storage location for the reference to an array.
- If a variable does not contain a reference to an array, the value of the variable is **null**.
- You cannot assign elements to an array unless it has already been created.



Creating Arrays

- After an array variable is declared, you can create an array by using the new operator and assign its reference to the variable with the following syntax:

```
arrayRefVar = new elementType[arraySize];
```

- This statement does two things
 - (1) it creates an array using **new elementType[arraySize]**;
 - (2) it assigns the reference of the newly created array to the variable **arrayRefVar**
- Example:

```
myList = new double[10];
```



Declaring and Creating in One Step

- Declaring an array variable, creating an array, and assigning the reference of the array to the variable can be combined in one statement as:

```
elementType[] arrayRefVar = new elementType[arraySize];
```

- or

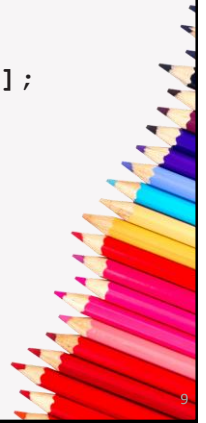
```
elementType arrayRefVar[] = new elementType[arraySize];
```

- Example:

```
• double[] myList = new double[10];
```

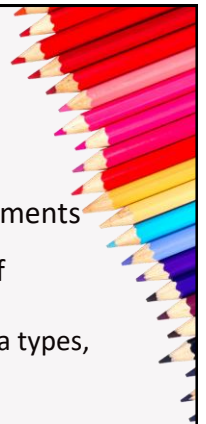
- To assign values to the elements, use the syntax:

```
• arrayRefVar[index] = value;
```



Array Size and Default Values

- Once an array is created, its size is fixed. It cannot be changed.
- You can find its size using `arrayRefVar.length`
- For example,
 - `myList.length` returns 10
- When an array is created, its elements are assigned the default value of
 - `0` for the numeric primitive data types,
 - `'\u0000'` for char types, and
 - `false` for boolean types.
 - `null` for reference types.



Accessing Array Elements

- The array elements are accessed through the index.
- Array indices are **0 based**; that is, they range from **0** to **arrayRefVar.length-1**.
- Each element in the array is represented using the following syntax, known as an *indexed variable*:
 - `arrayRefVar[index]` ;

- After an array is created, an indexed variable can be used in the same way as a regular variable.

- Examples:

```
myList[2] = myList[0] + myList[1];

for (int i = 0; i < myList.length; i++) {
    myList[i] = i;
}
```

Array Initializers

- Java has a shorthand notation, known as the *array initializer*, which combines the declaration, creation, and initialization of an array in one statement using the following syntax:

```
elementType[] arrayRefVar
    = {value0, value1, ...,
       valuek};
```

- Example:

```
double[] myList = {1.9,
    2.9, 3.4, 3.5};
```

- This shorthand notation is equivalent to the following statements:

```
double[] myList = new
    double[4];

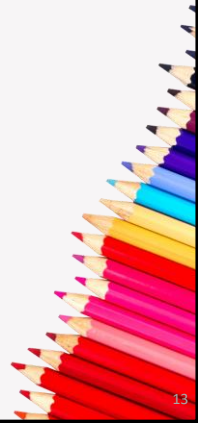
myList[0] = 1.9;
myList[1] = 2.9;
myList[2] = 3.4;
myList[3] = 3.5;
```

CAUTION

- The **new** operator is not used in the array-initializer syntax.
- Using an array initializer, you have to declare, create, and initialize the array **all in one statement**.
- Splitting it would cause a syntax error.

- For example, the following is wrong:

```
double[] myList;
myList = {1.9, 2.9, 3.4, 3.5};
```

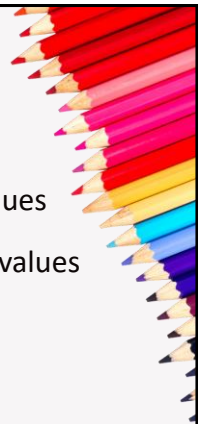


13

Processing Arrays

- When processing array elements, you will often use a **for loop**:
 1. All of the elements in an array are of the same type. They are evenly processed in the same fashion repeatedly using a loop.
 2. Since the size of the array is known, it is natural to use a for loop.
- Assume the array is created as follows:


```
double[] myList = new double[10];
```
- Examples of processing arrays:
 - Initializing arrays with input values
 - Initializing arrays with random values
 - Displaying/Printing arrays
 - Summing all elements
 - Finding the largest element
 - Finding the smallest index of the largest element
 - Random shuffling
 - Shifting elements



Initializing arrays with input values

- The following loop initializes the array **myList** with user input values.

```
java.util.Scanner input = new java.util.Scanner(System.in);
System.out.print("Enter " + myList.length + " values: ");
for (int i = 0; i < myList.length; i++)
    myList[i] = input.nextDouble();
```

Initializing arrays with random values

- The following loop initializes the array **myList** with random values between **0.0** and **100.0**, but less than **100.0**.

```
for (int i = 0; i < myList.length; i++) {
    myList[i] = Math.random() * 100;
}
```


Displaying arrays

- To print an array, you have to print each element in the array using a loop like the following:

```
for (int i = 0; i < myList.length; i++) {  
    System.out.print(myList[i] + " ");  
}
```

- For an array of the **char[]** type, it can be printed using one print statement. For example, the following code displays Dallas:

```
char[] city = {'D', 'a', 'l', 'l', 'a', 's'};  
System.out.println(city);
```

Summing all elements

- Use a variable named total to store the sum. Initially total is 0. Add each element in the array to total using a loop like this:

```
double total = 0;  
for (int i = 0; i < myList.length; i++) {  
    total += myList[i];  
}
```

Finding the largest element

- Use a variable named max to store the largest element.
- Initially max is myList[0]. To find the largest element in the array myList, compare each element with max, and update max if the element is greater than max.

```
double max = myList[0];  
for (int i = 1; i < myList.length; i++) {  
    if (myList[i] > max) max = myList[i];  
}
```

Finding the smallest index of the largest element

- Often you need to locate the largest element in an array. If an array has multiple elements with the same largest value, find the smallest index of such an element.

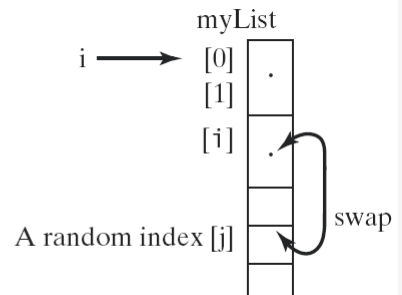
```
double max = myList[0];  
int indexOfMax = 0;  
for (int i = 1; i < myList.length; i++) {  
    if (myList[i] > max) {  
        max = myList[i];  
        indexOfMax = i;  
    }  
}
```

Random shuffling

- In many applications, you need to randomly reorder the elements in an array. This is called *shuffling*.
- To accomplish this, for each element **myList[i]**, randomly generate an index **j** and swap **myList[i]** with **myList[j]**, as follows:

```
for (int i = 0; i < myList.length - 1; i++) {
    // Generate an index j randomly
    int j = (int)(Math.random()
        * myList.length);

    // Swap myList[i] with myList[j]
    double temp = myList[i];
    myList[i] = myList[j];
    myList[j] = temp;
}
```



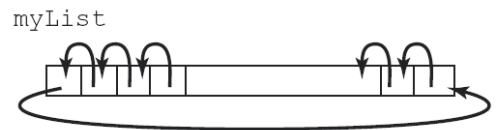
Shifting elements

- Sometimes you need to shift the elements left or right. Here is an example of shifting the elements one position to the left and filling the last element with the first element:

```
double temp = myList[0]; // Retain the first element
```

```
// Shift elements left
for (int i = 1; i < myList.length; i++) {
    myList[i - 1] = myList[i];
}
```

```
// Move the first element to fill in the last position
myList[myList.length - 1] = temp;
```



Simplifying coding

- Arrays can be used to greatly simplify coding for certain tasks.
- For example, suppose you wish to obtain the English name of a given month by its number. If the month names are stored in an array, the month name for a given month can be accessed simply via the index.
- The following code prompts the user to enter a month number and displays its month name:

```
String[] months = {"January", "February", ..., "December"};
System.out.print("Enter a month number (1 to 12): ");
int monthNumber = input.nextInt();
System.out.println("The month is " + months[monthNumber - 1]);
```

Foreach Loops(Enhanced for Loop)

- Java supports a convenient for loop, known as a *foreach loop*, which enables you to traverse the array sequentially without using an index variable.
- For example, the following code displays all the elements in the array myList:


```
for (double e: myList) {
    System.out.println(e);
}
```
- In general, the syntax for a foreach loop is


```
for (elementType element:
    arrayRefVar) {
    // Process the element
}
```
- You still have to use an index variable if you wish to traverse the array in a different order or change the elements in the array.

Case Study: Analyzing Numbers

- Read one hundred numbers, compute their average, and find out how many numbers are above the average.

AnalyzeNumbers

Run

Run with prepared input

25

Case Study: Deck of Cards

- The problem is to write a program that picks four cards randomly from a deck of 52 cards.
- All the cards can be represented using an array named deck, filled with initial values 0 to 51, as follows:

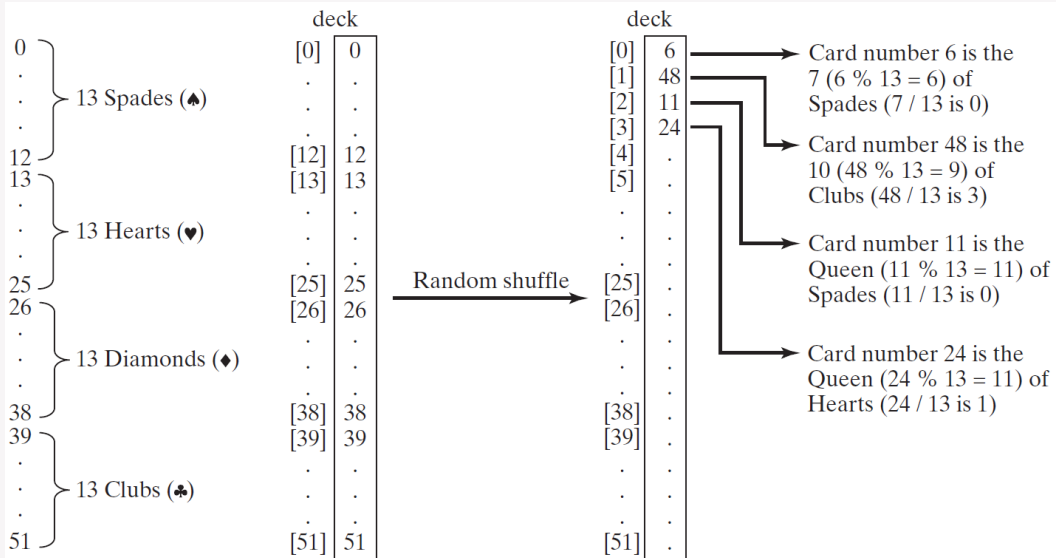
```
int[] deck = new int[52];
// Initialize cards
for (int i = 0; i < deck.length; i++)
    deck[i] = i;
```

DeckOfCards

Run

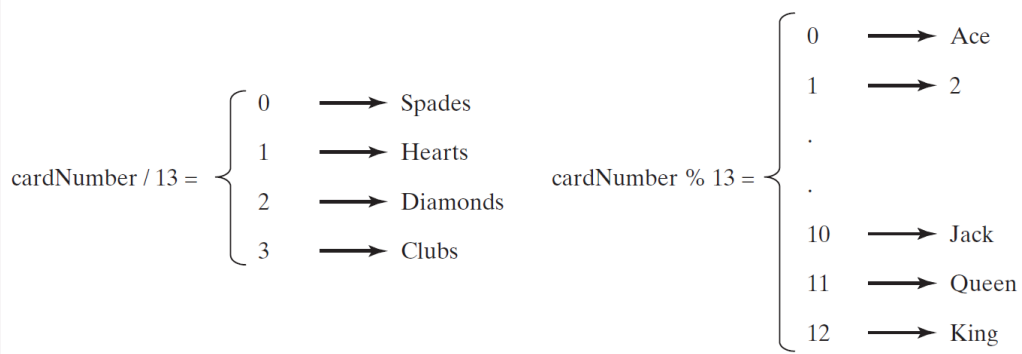
26

Case Study: Deck of Cards, cont.



27

Problem: Deck of Cards, cont.



DeckOfCards

Run

GUI Demo (picking four cards)

28

7.5 Copying Arrays

- Often, in a program, you need to duplicate an array or a part of an array. In such cases you could attempt to use the assignment statement (=), as follows:

```
list2 = list1;
```

Before the assignment:

```
list2 = list1;
```

list1 → Contents of list1

list2 → Contents of list2

After the assignment:

```
list2 = list1;
```

list1 → Contents of list1

list2 → Contents of list1

Contents of list2

29

Copying Arrays

- To copy the contents of one array into another, you have to copy the array's individual elements into the other array.
- There are three ways to copy arrays:

1. Use a **loop** to copy individual elements one by one.
2. Use the static **arraycopy method** in the **System** class.
3. Use the **clone method** to copy arrays; this will be introduced in Chapter 13, Abstract Classes and Interfaces.

```
int[] arrcopy = arr.clone();
```

30

Copying Arrays

- You can write a loop to copy every element from the source array to the corresponding element in the target array.
- Using a **for loop**:

```
int[] sourceArray = {2, 3, 1, 5, 10};  
int[] targetArray = new int[sourceArray.length];  
for (int i = 0; i < sourceArray.length; i++) {  
    targetArray[i] = sourceArray[i];  
}
```

Copying Arrays

- You can also use the **arraycopy** method in the **java.lang.System** class to copy arrays. The syntax for **arraycopy** is:

```
System.arraycopy(sourceArray, srcPos, targetArray,  
    tarPos, length);
```

- Example:

```
System.arraycopy(sourceArray, 0, targetArray, 0,  
    sourceArray.length);
```

- The arraycopy method does not allocate memory space for the target array. The target array must have already been created with its memory space allocated. After the copying takes place, targetArray and sourceArray have the same content but independent memory locations.

7.6 Passing Arrays to Methods

```
public static void printArray(int[] array) {  
    for (int i = 0; i < array.length; i++) {  
        System.out.print(array[i] + " ");  
    }  
}
```

//Invoke the method 1

```
int[] list = {3, 1, 2, 6, 4, 2};
```

```
printArray(list);
```

//Invoke the method 2

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

Anonymous Array

- The statement

```
printArray(new int[]{3, 1, 2, 6, 4, 2});
```

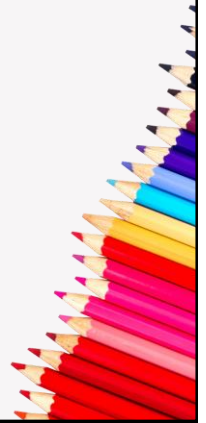
- creates an array using the following syntax:

```
new dataType[]{literal0, literal1, ...,  
    literalk};
```

- There is no explicit reference variable for the array. Such array is called an *anonymous array*.

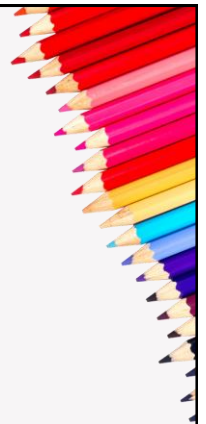
Passing Arrays to Methods

- Java uses *pass-by-value* to pass arguments to a method.
- 1. For an argument of a primitive type, the argument's value is passed.
- 2. For an argument of an array type, the value of the argument is a reference to an array; this reference value is passed to the method.
 - Semantically, it can be best described as *pass-by-sharing*, that is, the array in the method is the same as the array being passed.
 - Thus, if you change the array in the method, you will see the change outside the method.
- ***When passing an array to a method, the reference of the array is passed to the method.***



Pass By Value

- For a parameter of a **primitive type value**, the actual value is passed. Changing the value of the local parameter inside the method **does not affect** the value of the variable outside the method.
- For a parameter of **an array type**, the value of the parameter contains a reference to an array; this reference is passed to the method. Any changes to the array that occur inside the method body **will affect** the original array that was passed as the argument.



Simple Example

```
public class TestArrayArguments {
    public static void main(String[] args) {
        int x = 1; // x represents an int value
        int[] y = new int[10]; // y represents an array of int values

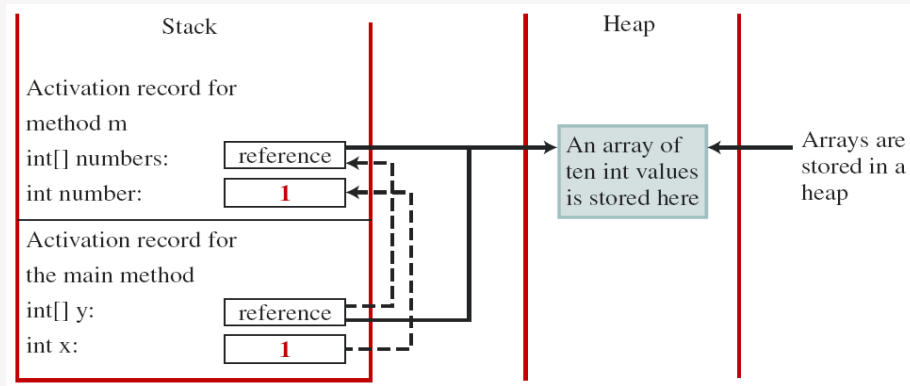
        m(x, y); // Invoke m with arguments x and y

        System.out.println("x is " + x);
        System.out.println("y[0] is " + y[0]);
    }

    public static void m(int number, int[] numbers) {
        number = 1001; // Assign a new value to number
        numbers[0] = 5555; // Assign a new value to numbers[0]
    }
}
```

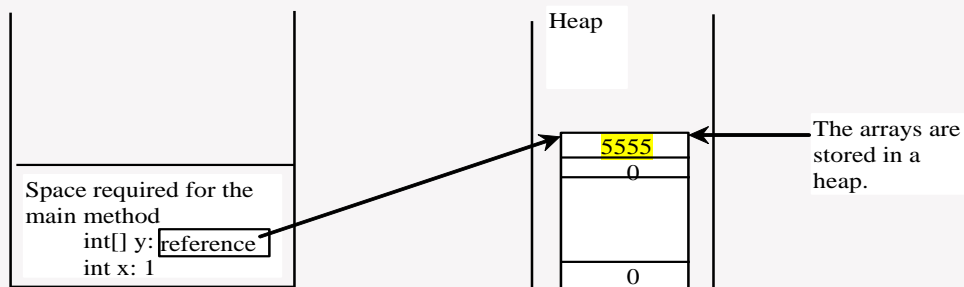
Call Stack

- When invoking `m(x, y)`, the values of `x` and `y` are passed to `number` and `numbers`. Since `y` contains the reference value to the array, `numbers` now contains the same reference value to the same array.



Heap

- The JVM stores the array in an area of memory, called heap, which is used for dynamic memory allocation where blocks of memory are allocated and freed in an arbitrary order.



39

Passing Arrays as Arguments

- Objective: Demonstrate differences of passing primitive data type variables and array variables.

```

/** Swap two variables */
public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
}

/** Swap the first two elements in the array */
public static void swapFirstTwoInArray(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
}

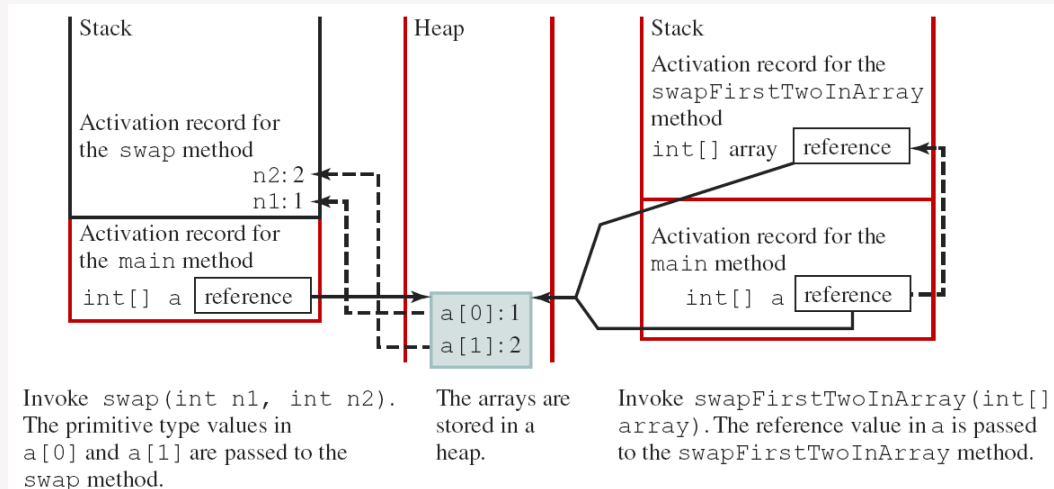
```

[TestPassArray](#)

Run

40

Call Stack



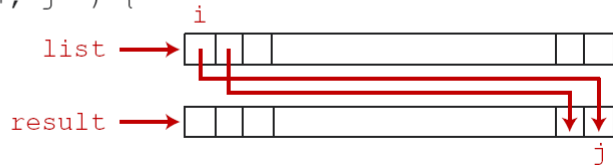
7.7 Returning an Array from a Method

- When a method returns an array, the reference of the array is returned.

```

1 public static int[] reverse(int[] list) {
2     int[] result = new int[list.length];
3
4     for (int i = 0, j = result.length - 1;
5         i < list.length; i++, j--) {
6         result[j] = list[i];
7     }
8
9     return result;
10 }

```



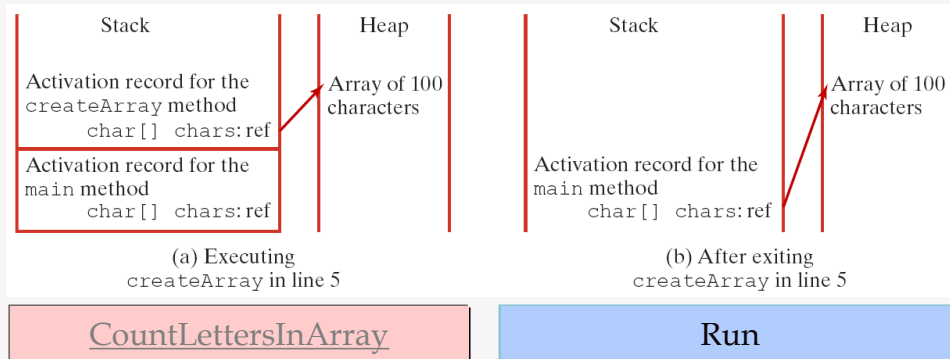
```

int[] list1 = {1, 2, 3, 4, 5, 6};
int[] list2 = reverse(list1);

```

Case Study: Counting Occurrence of Each Letter

- Generate 100 lowercase letters randomly and assign to an array of characters.
- Count the occurrence of each letter in the array.



43

7.9 Variable-Length Argument Lists

- You can pass a variable number of arguments of the same type to a method. The parameter in the method is declared as follows:
typeName... parameterName
- In the method declaration, you specify the type followed by an ellipsis (...).
 - Only one variable-length parameter may be specified in a method, and this parameter must be the last parameter. Any regular parameters must precede it.
- *A variable number of arguments of the same type can be passed to a method and treated as an array.*


Example

```
1 public class VarArgsDemo {
2     public static void main(String[] args) {
3         printMax(34, 3, 3, 2, 56.5);
4         printMax(new double[]{1, 2, 3});
5     }
6
7     public static void printMax(double... numbers) {
8         if (numbers.length == 0) {
9             System.out.println("No argument passed");
10            return;
11        }
12
13        double result = numbers[0];
14
15        for (int i = 1; i < numbers.length; i++)
16            if (numbers[i] > result)
17                result = numbers[i];
18
19        System.out.println("The max value is " + result);
20    }
21 }
```


VarArgsDemo

Run

7.10 Searching Arrays

- Searching is the process of looking for a specific element in an array.
 - There are many algorithms and data structures devoted to searching. In this section, two commonly used approaches are discussed, **linear search** and **binary search**.
 - *If an array is sorted, binary search is more efficient than linear search for finding an element in the array.*
- 

Linear Search

- The linear search approach compares the key element, key, sequentially with each element in the array list. The method continues to do so until the key matches an element in the list or the list is exhausted without a match being found. If a match is made, the linear search returns the index of the element in the array that matches the key. If no match is found, the search returns -1.

```

1 public class LinearSearch {
2     /** The method for finding a key in the list */
3     public static int linearSearch(int[] list, int key) {
4         for (int i = 0; i < list.length; i++) {
5             if (key == list[i])
6                 return i;
7         }
8         return -1;
9     }
10 }

```

list [0] [1] [2] ...
 key Compare key with list[i] for i = 0, 1, ...

47

animation

Linear Search Animation

Key

List

3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8
3	6	4	1	9	7	3	2	8

48

Binary Search

- For binary search to work, the elements in the array must already be ordered.
 - Without loss of generality, assume that the array is in ascending order.
 - e.g., 2 4 7 10 11 45 50 59 60 66 69 70 79
 - The binary search first compares the key with the element in the middle of the array.
- If the key is less than the middle element, you only need to search the key in the first half of the array.
 - If the key is equal to the the middle element, the search ends with a match.
 - If the key is greater than the middle element, you only need to search the key in the second half of the array.

49

animation

Binary Search Animation

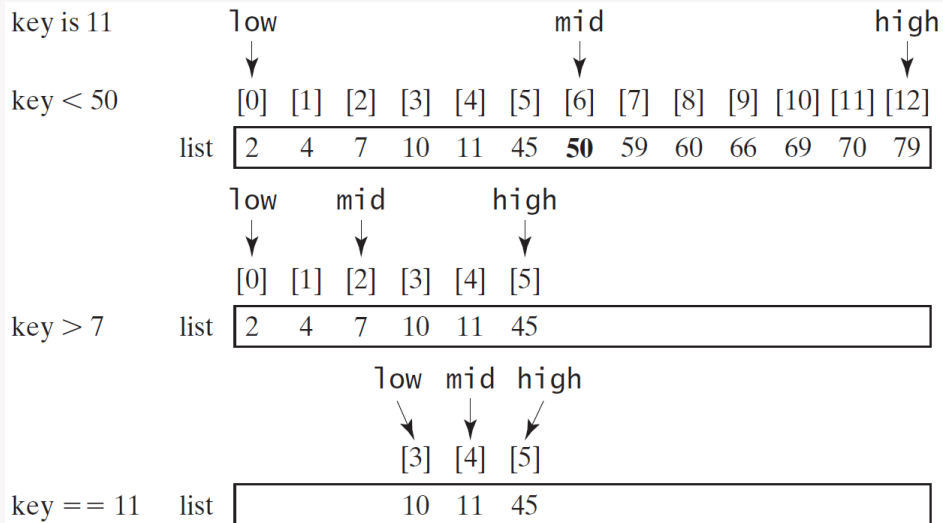
Key

List

8	1	2	3	4	6	7	8	9
8	1	2	3	4	6	7	8	9
8	1	2	3	4	6	7	8	9

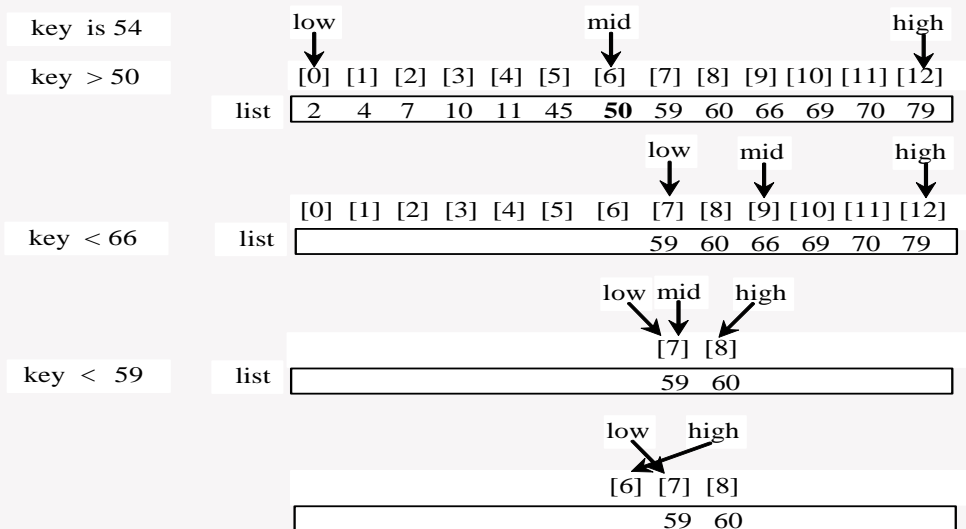
50

Binary Search, cont.



51

Binary Search, cont.



52

Binary Search, cont.

- The `binarySearch` method returns the index of the element in the list that matches the search key if it is contained in the list. Otherwise, it returns

`-insertion point - 1.`

- The insertion point is the point at which the key would be inserted into the list.

53

From Idea to Solution

```

1 public class BinarySearch {
2     /** Use binary search to find the key in the list */
3     public static int binarySearch(int[] list, int key) {
4         int low = 0;
5         int high = list.length - 1;
6
7         while (high >= low) {
8             int mid = (low + high) / 2;
9             if (key < list[mid])
10                 high = mid - 1;
11             else if (key == list[mid])
12                 return mid;
13             else
14                 low = mid + 1;
15         }
16
17         return -low - 1; // Now high < low, key not found
18     }
19 }

```

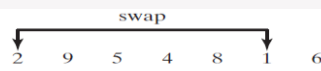
54

7.11 Sorting Arrays

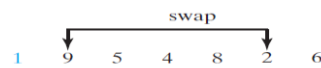
- Sorting, like searching, is also a common task in computer programming.
- Many different algorithms have been developed for sorting.
- This section introduces a simple, intuitive sorting algorithms: *selection sort*.
- Selection sort finds the smallest number in the list and places it first. It then finds the smallest number remaining and places it second, and so on until the list contains only a single number.

Selection Sort

Select 1 (the smallest) and swap it with 2 (the first) in the list.

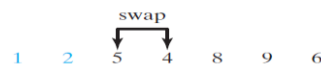


The number 1 is now in the correct position and thus no longer needs to be considered.



Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.

The number 2 is now in the correct position and thus no longer needs to be considered.



Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.

The number 4 is now in the correct position and thus no longer needs to be considered.



5 is the smallest and in the right position. No swap is necessary.

The number 5 is now in the correct position and thus no longer needs to be considered.



Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.

The number 6 is now in the correct position and thus no longer needs to be considered.



Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.

The number 8 is now in the correct position and thus no longer needs to be considered.



Since there is only one element remaining in the list, the sort is completed.

From Idea to Solution

- The solution can be described as follows:

```
for (int i = 0; i < list.length - 1; i++) {
    select the smallest element in list[i..list.length-1];
    swap the smallest with list[i], if necessary;
    // list[i] is in its correct position.
    // The next iteration applies on list[i+1..list.length-1]
}
```

List 7.8

```
1 public class SelectionSort {
2     /** The method for sorting the numbers */
3     public static void selectionSort(double[] list) {
4         for (int i = 0; i < list.length - 1; i++) {
5             // Find the minimum in the list[i..list.length-1]
6             double currentMin = list[i];
7             int currentMinIndex = i;
8
9             for (int j = i + 1; j < list.length; j++) {
10                 if (currentMin > list[j]) {
11                     currentMin = list[j];
12                     currentMinIndex = j;
13                 }
14             }
15
16             // Swap list[i] with list[currentMinIndex] if necessary
17             if (currentMinIndex != i) {
18                 list[currentMinIndex] = list[i];
19                 list[i] = currentMin;
20             }
21         }
22     }
23 }
```

7.12 The Arrays Class

- The **java.util.Arrays** class contains various static methods for sorting and searching arrays, comparing arrays, filling array elements, and returning a string representation of the array.
- These methods are overloaded for all primitive types.
 - **sort** , **parallelSort**
 - **binarySearch**
 - **equals**
 - **fill**
 - **toString**

The Arrays.sort Method

- You can use the **sort** or **parallelSort** method to sort a whole array or a partial array.

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};  
java.util.Arrays.sort(numbers); // Sort the whole array  
java.util.Arrays.parallelSort(numbers); // Sort the whole array  
  
char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};  
java.util.Arrays.sort(chars, 1, 3); // Sort part of the array  
java.util.Arrays.parallelSort(chars, 1, 3); // Sort part of the array
```

The Arrays.binarySearch Method

- You can use the **binarySearch** method to search for a key in an array. The array must be presorted in increasing order. If the key is not in the array, the method returns **-(insertionIndex+ 1)**.

```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("1. Index is " +
    java.util.Arrays.binarySearch(list, 11));
System.out.println("2. Index is " +
    java.util.Arrays.binarySearch(list, 12));

char[] chars = {'a', 'c', 'g', 'x', 'y', 'z'};
System.out.println("3. Index is " +
    java.util.Arrays.binarySearch(chars, 'a'));
System.out.println("4. Index is " +
    java.util.Arrays.binarySearch(chars, 't'));
```

Index is 4
Index is -6
Index is 0
Index is -4

61

The Arrays.equals Method

- You can use the **equals** method to check whether two arrays are strictly equal.
- Two arrays are strictly equal if their corresponding elements are the same.

```
int[] list1 = {2, 4, 7, 10};
int[] list2 = {2, 4, 7, 10};
int[] list3 = {4, 2, 7, 10};
System.out.println(java.util.Arrays.equals(list1, list2)); // true
System.out.println(java.util.Arrays.equals(list2, list3)); // false
```

The `Arrays.fill` Method

- You can use the **fill** method to fill in all or part of the array.

```
int[] list1 = {2, 4, 7, 10};  
int[] list2 = {2, 4, 7, 7, 7, 10};  
java.util.Arrays.fill(list1, 5); // Fill 5 to the whole array  
java.util.Arrays.fill(list2, 1, 5, 8); // Fill 8 to a partial array
```

The `Arrays.toString` Method

- You can use the **toString** method to return a string that represents all elements in the array.
- This is a quick and simple way to display all elements in the array.

```
int[] list = {2, 4, 7, 10};  
System.out.println(java.util.Arrays.toString(list));
```

```
[2, 4, 7, 10]
```


Pass Arguments to Invoke the Main Method

- A main method is just a regular method.
- Can you pass arguments to main? Of course, yes.
- In the following examples, the main method in class TestMain is invoked by a method in A.

```
public class A {
    public static void main(String[] args) {
        String[] strings = {"New York",
                           "Boston", "Atlanta"};
        TestMain.main(strings);
    }
}
```

```
public class TestMain {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

7.13 Command-Line Parameters

- *The main method can receive string arguments from the command line.*

```
public class TestMain {
    public static void main(String[] args) {
        for (int i = 0; i < args.length; i++)
            System.out.println(args[i]);
    }
}
```

```
java TestMain arg0 arg1 arg2
```

```
java TestMain "First num" alpha 53
```

- When the main method is invoked, the Java interpreter creates an array to hold the command-line arguments and pass the array reference to args. The Java interpreter then passes args to invoke the main method.
- If you run the program with no strings passed, the array is created with new String[0]. In this case, the array is empty with length 0. args references to this empty array. Therefore, args is not null, but args.length is 0.

Case Study: Calculator

- Objective: Write a program that will perform binary operations on integers.
- The program receives three parameters: an operator and two integers.

```
java Calculator 2 + 3
```

```
java Calculator 2 - 3
```

```
java Calculator 2 / 3
```

```
java Calculator 2 . 3
```

Calculator

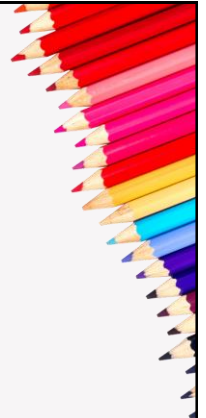
Run

67

Chapter Summary

Chapter Summary

- A variable is declared as an array type using the syntax `elementType[] arrayRefVar` or `elementType arrayRefVar[]`.
- The declaration of an array variable does not allocate any space in memory for the array. An array variable contains a reference to an array.
- You can create an array by using the new operator with the following syntax: `new elementType[arraySize]`.
- After an array is created, its size becomes permanent and can be obtained using `arrayRefVar.length`.



Chapter Summary

- Each element in the array is represented using the syntax `arrayRefVar[index]`. An out-of-bounds error will occur if you attempt to reference elements beyond the bounds of an array.
- When an array is created, its elements are assigned the default value of 0 for the numeric primitive data types, `\u0000` for char types, and `false` for boolean types.
- When you pass an array argument to a method, you are actually passing the reference of the array; that is, the called method can modify the elements in the caller's original array.



Programming Exercises

*3, 5, 7, 9, 10, 13, 17,
18, 26, 27, 29, 34, 35*

