BIRZEIT UNIVERSITY

Arrays

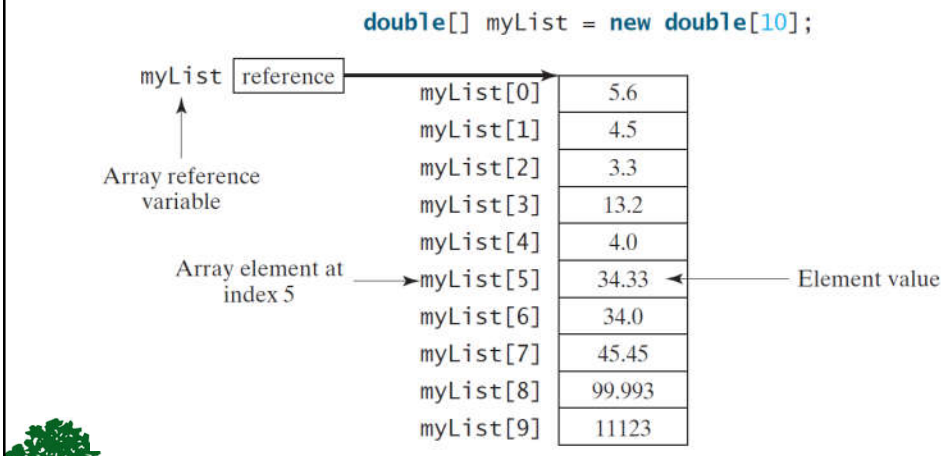
By: Mamoun Nawahdah (Ph.D.)
2022

Opening Problem

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

Introducing Arrays

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



3

Declaring Array Variables

datatype[] arrayRefVar;

Example:

double[] myList;

datatype arrayRefVar**[]**; // This style is allowed, but **not preferred**

Example:

double myList[];

4

Creating Arrays

arrayRefVar = new datatype[arraySize];

Example:

myList = new double[10];

- **myList[0]** references the 1st element.
- **myList[9]** references the last element.

5

Declaring and Creating in 1 Step

datatype[] arrayRefVar = new datatype[arraySize];

double[] myList = new double[10];

datatype arrayRefVar[] = new datatype[arraySize];

double myList[] = new double[10];

6

The Length of an Array

- 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

7

Default Values

- When an array is created, its elements are assigned the **default value** of :
  - **0** for the numeric data types.
  - '**\u0000**' for **char** types.
  - **false** for **boolean** types.

8

Indexed Variables

❖ The array elements are accessed through the **index**.

❖ The array indices are ***0-based***, i.e., it starts from **0** to **arrayRefVar.length-1**.

❖ Each element in the array is represented using the following syntax, known as an *indexed variable*:

arrayRefVar[index];

9

Using Indexed Variables

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

❖ For example, the following code adds the value in **myList[0]** and **myList[1]** to **myList[2]**:

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

10

Array Initializers

❖ Declaring, creating, initializing in **1 step**:

```
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;
```

11

CAUTION

❖ Using the shorthand notation, 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};
```

Trace Program with Arrays

```
public class Test {
  public static void main(String[] args) {
      int[] values = new int[5];
      for (int i = 1; i < values.length; i++) {
          values[i] = i + values[i-1];
      }
      values[0] = values[1] + values[4];
  }
}
```

0	11
1	1
2	3
3	6
4	10

13

Initializing with input values

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

Scanner input = new Scanner(System.in);

System.out.print("Enter " + myList.length + " values: ");

for (int i = 0 ; i < myList.length ; i++)

      myList[ i ] = input.nextDouble();
```

14

Initializing with random values

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

Printing arrays

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

15

Summing all elements

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

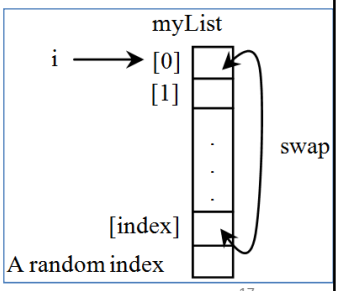
Finding the largest element

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

16

Random Shuffling

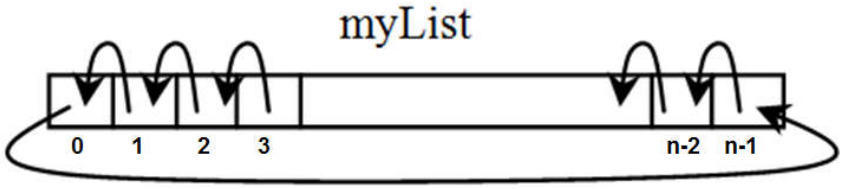
```
for (int i = 0; i < myList.length; i++) {
  // Generate an index j randomly
  int index = (int)(Math.random()
    * myList.length);

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



17

Shifting Elements

```
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;
```



18

Enhanced for Loop (for-each loop)

❖ **A** new **for** loop that enables you to traverse the complete array sequentially without using an index variable.

- For example, the following code displays all elements in the array **myList**:

```
for (double value : myList)
    System.out.println(value);
```

❖ In general, the syntax is:

```
for (elementType value : arrayRefVar) {
    // Process the value
}
```

❖ 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.

19

Analyze Numbers

Write a program to read one hundred numbers, compute their average, and find out how many numbers are above the average?

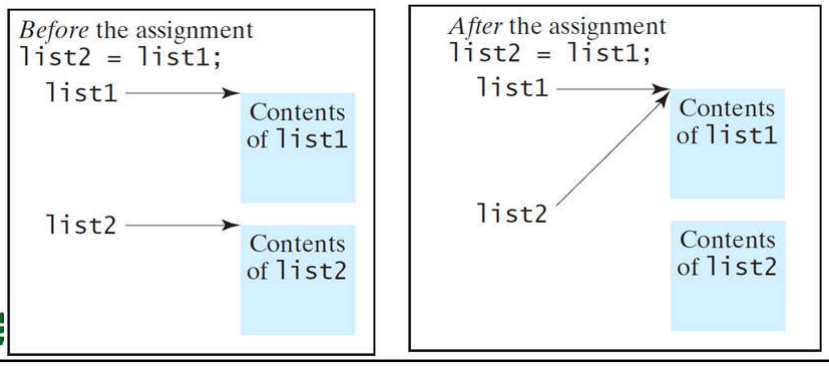
20

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;



Copying Arrays

❖ **Using a loop:**

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
    targetArray[i] = sourceArray[i];
```

22

The arraycopy Utility

System.arraycopy(sourceArray, src_pos, targetArray, tar_pos, length);

❖ Example:

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

23

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**

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

printArray(list);

24

Anonymous Array

- The statement

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

- Creates array using the following syntax:

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

- There is **no explicit** reference variable for the created array.

- Such array is called an ***anonymous array***.

25

Pass by Value/Reference

- 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.

26

Example: Pass by value/reference

```
public class Test {
    public static void main(String[] args) {
        int x = 1;
        int[] y = {1,2,3,4,5};
        m(x, y);
        System.out.println(" x is " + x );
        System.out.println(" y[0] is " + y[0] );
    }

    static void m( int number,   int[] numbers ) {
        number = 1001;
        numbers[0] = 5005;
    }
}
```

27

Returning an Array from a Method

```
public static int[] reverse(int[] list) {
  int[] result = new int[list.length];
  for (int i=0, j=result.length - 1; i < list.length; i++, j--) {
        result[j] = list[i];
  }
  return result;
}
```

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

28

Problem: 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.

29

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**.

30

From Idea to Solution

```
public static int linearSearch(int[] list, int key) {
    for (int i = 0; i < list.length; i++)
        if (key == list[i])     return i;
    return -1;
}
```

Trace the method:

```
int[] list = {1, 4, 4, 2, 5, -3, 6, 2};
int i = linearSearch(list, 4);  // returns 1
int j = linearSearch(list, -4); // returns -1
int k = linearSearch(list, -3); // returns 5
```

31

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.

32

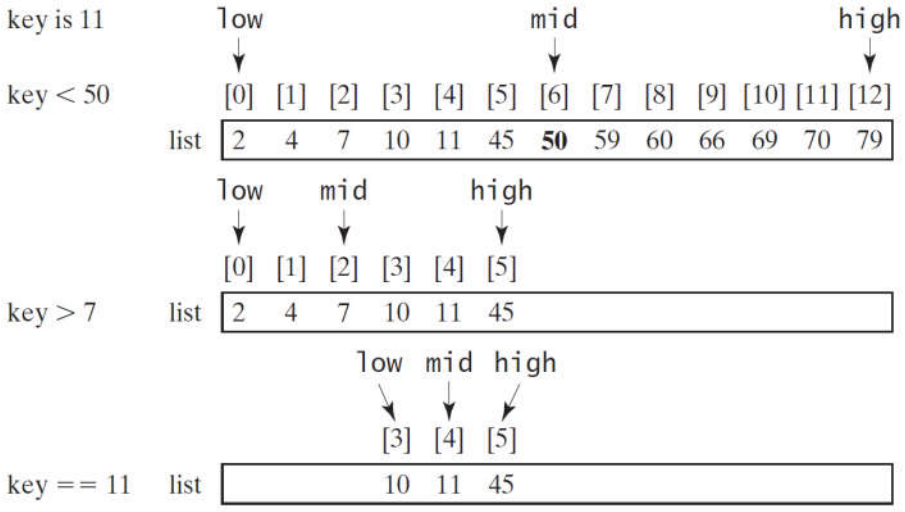
Binary Search, cont.

Consider the following three cases:

- 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 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.

33

Binary Search, cont.



34

From Idea to Solution

```
public static int binarySearch( int[] list,  int key ) {
        int low = 0;
        int high = list.length - 1;
        while (high >= low) {
                int mid = (low + high) / 2;
                if (key < list[mid])
                        high = mid - 1;
                else if (key == list[mid])
                        return mid;
                else
                        low = mid + 1;
        }
        return -1 - low;
}
```

35

The Arrays.binarySearch Method

- Since binary search is frequently used in programming, Java provides several **binarySearch** methods for searching a key in an array of **int**, **double**, **char**, **short**, **long**, and **float** in the **java.util.Arrays** class.

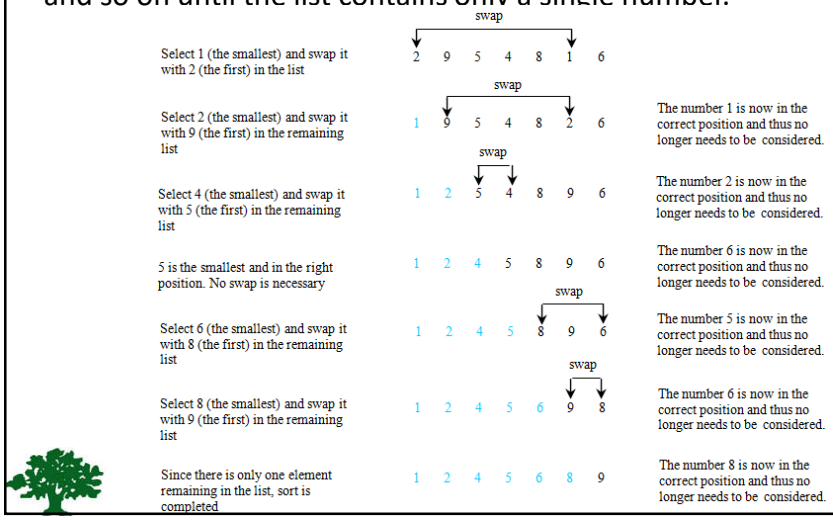
```
int[] list = {2, 4, 7, 10, 11, 45, 50, 59, 60, 66, 69, 70, 79};
System.out.println("Index is " +  Arrays.binarySearch(list, 11));

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

- For the **binarySearch** method to work, the array **must be pre-sorted** in increasing order.

36

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.



37

From Idea to Solution

- Selection sort algorithm:

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

38

Wrap it in a Method

```
public static void selectionSort( double[] list) {
   for (int i = 0; i < list.length; i++) {
     // Find the minimum in the list[i..list.length-1]
     double currentMin = list[i];
     int currentMinIndex = i;
     for (int j = i + 1; j < list.length; j++) {
          if (currentMin > list[j]) {
                currentMin = list[j];
                currentMinIndex = j;
          }
     }
     // Swap list[i] with list[currentMinIndex] if necessary;
     if (currentMinIndex != i) {
          list[currentMinIndex] = list[i];
          list[i] = currentMin;
     }
   }
}
```

39

The Arrays.sort Method

- Java provides several sort methods for sorting an array of **int**, **double**, **char**, **short**, **long**, and **float** in the **java.util.Arrays** class.
- For example, the following code sorts an array of numbers and an array of characters:

```
double[] numbers = {6.0, 4.4, 1.9, 2.9, 3.4, 3.5};
java.util.Arrays.sort(numbers);

char[] chars = {'a', 'A', '4', 'F', 'D', 'P'};
java.util.Arrays.sort(chars);
```

40

main Method is just a Regular Method

- You can call a regular method by passing actual parameters.
- You can pass **arguments** to **main**.
- For example, the main method in class **B** is invoked by a method in **A**, as shown below:

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

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

Command-Line Parameters

```
class TestMain {
  public static void main(String[] s) {
    ...
  }
}
```

```
java TestMain arg0 arg1 arg2 ... argn
```

- In the **main** method, get the arguments from **s[0], s[1], ..., s[n]**, which corresponds to **arg0, arg1, ..., argn** in the command line.

42

Problem: Calculator

- **Objective**: Write a class "**Calculator**" that will perform binary operations on integers.
- The class receives three parameters: an **operator** and **two integers** as follow:

java Calculator 2 + 3
java Calculator 2 - 3
java Calculator 2 / 3
java Calculator 2 . 3

43

Declare/Create 2D Arrays

```
// Declare array refvar
dataType[][] refVar;

// Create array and assign its reference to variable
refVar = new dataType[10][10];

// Combine declaration and creation in one statement
dataType[][] refVar = new dataType[10][10];

// Alternative syntax
dataType refVar[][] = new dataType[10][10];
```

44

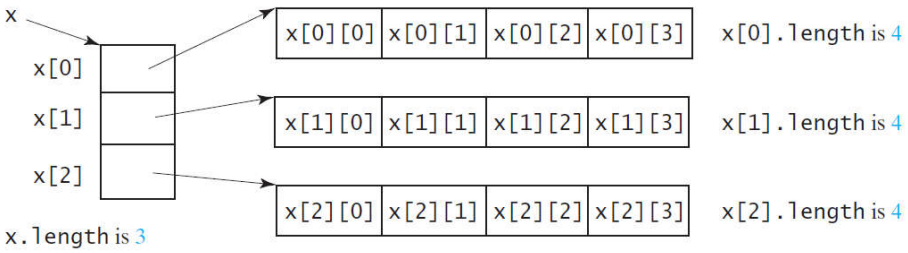
Creating 2D Arrays

```
int[][] matrix = new int[3][2];

for (int i = 0; i < matrix.length; i++)
  for (int j = 0; j < matrix[i].length; j++)
    matrix[i][j] = (int)(Math.random() * 100);
```

45

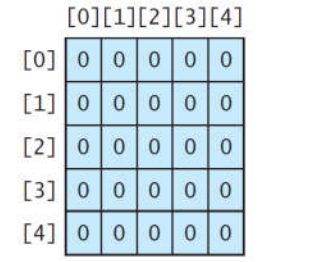
Lengths of 2D Arrays

int[][] x = new int[3][4];

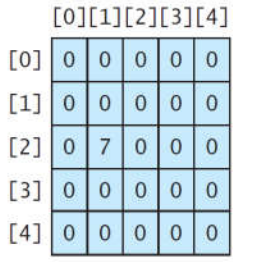


46

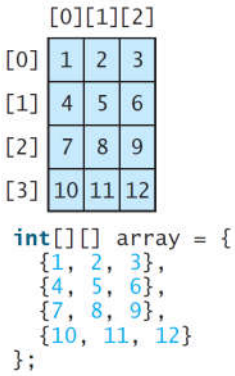
Two-dimensional Array Illustration



(a)



(b)



(c)

matrix.length? 5

matrix[0].length? 5

array.length? 4

array[0].length? 3

47

Lengths of 2D Arrays, cont.

```
int[][] array = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9},
        {10, 11, 12}
};
```

array.length
array[0].length
array[1].length
array[2].length
array[3].length

array[4].length → **ArrayIndexOutOfBoundsException**

48

Declaring, Creating, and Initializing Using Shorthand Notations

- You can also use an array **initializer** to declare, create and initialize a 2-dimensional array.
- For example:

```
int[][] array = {
  {1, 2, 3} ,
  {4, 5, 6} ,
  {7, 8, 9} ,
  {10, 11, 12}
};
```

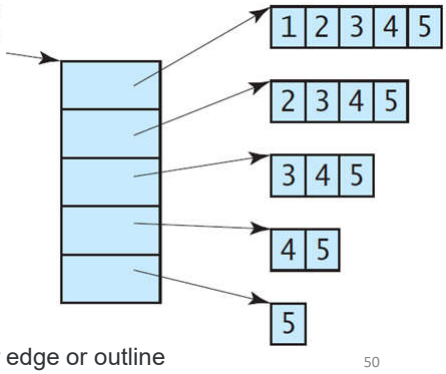
49

Ragged Arrays

- Each row in a 2D array is **itself** an array. So, the **rows can have different lengths**.
- Such an array is known as *a **ragged array***.

For example:

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



Ragged: having an irregular edge or outline

50

Initializing with input values

```
java.util.Scanner input = new Scanner(System.in);
System.out.println("Enter "+ arr.length + " rows and "
      +  arr[0].length + " columns: ");
for (int r = 0; r < arr.length; r++) {
  for (int c = 0; c < arr[r].length; c++) {
    arr[r][c] = input.nextInt();
  }
}
```

51

Printing Arrays

```
for (int r = 0; r < arr.length; r++) {
    for (int c = 0; c < arr[r].length; c++)
            System.out.print(arr[r][c] + " ");
    System.out.println();
}
```

52

Summing all elements

```
int total = 0;
for (int r = 0; r < arr.length; r++) {
  for (int c = 0; c < arr[r].length; c++) {
        total += arr[r][c];
  }
}
```

53

Summing elements by column

```
for (int c = 0; c < arr[0].length; c++) {
  int total = 0;
  for (int r = 0; r < arr.length; r++)
      total += arr[r][c];
  System.out.println("Sum for column " + c + " is " + total);
}
```

54

Multidimensional Arrays

❖ Occasionally, you will need to represent

n-dimensional data structures.

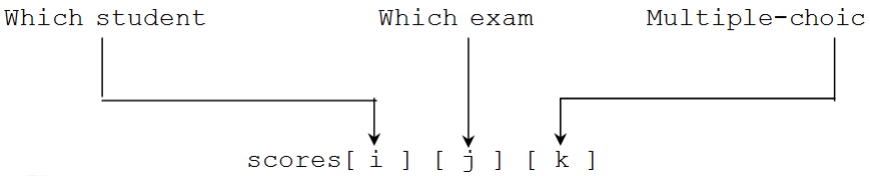
❖ In Java, you can create n-dimensional arrays for any integer **n**.

❖ The way to declare two-dimensional array variables and create **2**-dimensional arrays can be generalized to declare **n**-dimensional array variables and create **n**-dimensional arrays for **n > 2**.

55

Multidimensional Arrays

```
double[][][] scores = {
  {{7.5, 20.5}, {9.0, 22.5}, {15, 33.5}, {13, 21.5}, {15, 2.5}},
  {{4.5, 21.5}, {9.0, 22.5}, {15, 34.5}, {12, 20.5}, {14, 9.5}},
  {{6.5, 30.5}, {9.4, 10.5}, {11, 33.5}, {11, 23.5}, {10, 2.5}},
  {{6.5, 23.5}, {9.4, 32.5}, {13, 34.5}, {11, 20.5}, {16, 7.5}},
  {{8.5, 26.5}, {9.4, 52.5}, {13, 36.5}, {13, 24.5}, {16, 2.5}},
  {{9.5, 20.5}, {9.4, 42.5}, {13, 31.5}, {12, 20.5}, {16, 6.5}}};
```



56