Arrays

Chap. 9

Storing Collections of Values

Introductory Example

Problem: Teachers need to be able to compute a variety of grading statistics for student scores. (E. g., Mrs. White in §9.1)

One Solution: Develop an assignment class that represents student scores on an assignment (e.g., a test or homework) and is able to print statistics for that assignment.

Preliminary Analysis

Different statistics require different capabilities:

- Average (arithmetic mean), max, min, range:
  Only one pass through the data needed to calculate these.
- Deviations (differences) from the mean, standard deviation:
  Two passes needed— one to find average, one to find deviations.
- Median (requires sorting), mode:
  Multiple passes through the data needed.

How to proceed:

- Option 1: Enter the data as many times as necessary.
  Yuck! There must be a better way!
- Option 2: Put the data in a file on disk and read from the file as many times as necessary.
  Yes, this would work, but file I/O is slow (and we don't know how to do file I/O yet).
- Option 3: Store the data in an in-memory container:
  - an array
  - a `Vector`
  - an `ArrayList`
  - a `LinkedList`

Behavior

Our program should read the list of student names and assignment scores. It should then compute and display the average score and then the list of student names along with their deviations from the average.

5

Algorithm for program

If we have an Assignment class that reads and stores the student names and scores, calculates the statistics (average and deviations), and displays the student names and statistics,the main program is easy:

1. Construct *Keyboard* object *theKeyboard*, Screen object *theScreen*, and *Assignment* object *theAssignment*.
2. Ask *theAssignment* to read the student info, displaying prompts on *theScreen* and reading input from *theKeyboard*.
3. Ask *theAssignment* to display the student names and statistics on *theScreen*.

6

Building the `Assignment` Class

Behavior (operations):

- Constructor
- read()
- average()
- printStats()

Attributes (data):

- class size
- storage (array of strings) for names
- storage (array of doubles) for scores

7

The Code

```
import ann.easyio.*;
class Assignment {
  public Assignment()  {
    studentNames = null;
    studentScores = null;
    size = 0;
  }
  public double average()  {
    int sum = 0;
    for (int i = 0; i < size; i++)
      sum += studentScores[i];
    return (double)sum / size;
  }
  public void printStats(Screen theScreen)  {
    double theAverage = average();
    theScreen.println("\nThe average is: " + average());
    theScreen.println("The standard deviations are:");
    for (int i = 0; i < size; i++)
      theScreen.println(studentNames[i] + " "
                        + studentScores[i] + " " +
                        + (studentScores[i] - theAverage) );
  }
```

1

8

```
  public void read(Screen theScreen, Keyboard theKeyboard) {
    theScreen.print("Enter the size of the class: ");
    size = theKeyboard.readInt();
    if (size <= 0) {
      theScreen.println("invalid Assignment size: " + size);
      System.exit(-1);
    }
    else {
      studentNames = new String [size];
      studentScores = new double [size];
    }

    String name;
    theScreen.println("Enter the names and scores of "+
                      "the students in the class: ");
    for (int i = 0; i < size; i++) {
      theScreen.print((i + 1) + ": ");
      Keyboard.EatWhiteSpace();
      studentNames[i] = (theKeyboard.readLine());
      studentScores[i] = theKeyboard.readInt();
     }
  }

  private int size;
  private String [] studentNames;
  private double [] studentScores;

} // end of class Assignment
```

2

9

```
class Teacher {
  public static void main(String [] args)  {
    Screen theScreen = new Screen();
    Keyboard theKeyboard = new Keyboard();
    Assignment theAssignment = new Assignment();
    theAssignment.read(theScreen, theKeyboard);
    theAssignment.printStats(theScreen);
  }
}
```

10

Sample run:

```
Enter the size of the class: 3
Enter the names and scores of the students in the class:
1: Joe Blow
80
2: L. Nyhoff
100
3: Mary Doe
90

The average is: 90.0
The standard deviations are:
Joe Blow 80 -10.0
L. Nyhoff 100 10.0
Mary Doe 90 0.0
```

2

11

Array Definitions

Java arrays are objects

- They must be accessed via handles

Three declaration forms:

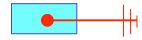
1. Uninitialized:

Type `[]` *arrayName;*

Example:

```
double [] array1;
```


```
array1 = null;
```



Later **`array1`** can be set to **`null`** or assigned (the address of) an array created with **`new`** — e.g., see **`Assignment`**.

12

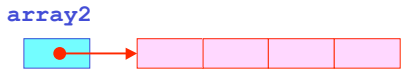
2. Using **new** operator

```
Type [] arrayName =
        new Type [size];
```

where ***size*** is an integer-valued expression that specifies the number of elements in the array.

Example:

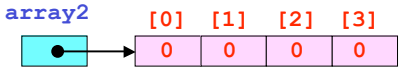
```
int size = theKeyboard.readInt(); // e.g, 4
int [] array2 = new int [size];
```



13

Notes:

- A block of memory for the array is allocated during execution.
- The address of this memory block is stored in the array variable; so an array variable is a handle.
- Array elements are initialized with default values
  - 0 for numeric types
  - **false** for boolean
  - **null** for reference type
- Array indexes are numbered beginning with 0.



14

3. Initialize with an array literal:

```
Type [] arrayName = array-literal;
```

where ***array-literal*** is a list of values of type ***Type*** enclosed in curly braces **{}** and separated by commas.

Example:

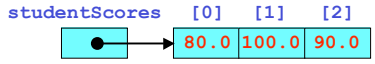
```
double [] array3 = {0, 3.7, 2.25};
```

A block of memory of appropriate size is allocated and initialized with the listed values; address is stored in **array3**.

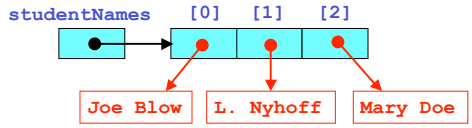
15

Array Properties

- Arrays are zero-based: array indexes are numbered beginning with 0.
- Elements of arrays of primitive types are stored as values; e.g. in **Assignment**:



- Elements of arrays of reference types are stored as handles; e.g. in **Assignment**:



16

- The latest version of Java allows the use of array literals in the first form of array declaration

```
Type [] arrayName =
          new Type [] array-literal;
```

and also allows this expression to be used to assign a value to an array variable:

```
arrayName = new Type [] array-literal;
```

Example:

```
double [] array3 =
              new double [] {0, 3.7, 2.25};
double [] array4;
   . . .
array4 = new double [] {1.1, 2.2, 3.3, 4.4};
```

17

- Each array has a *public* attribute **`length`** whose value is the number of array elements.
  Example: For **`studentScores`** in our sample run, **`studentScores.length`** is 3.
- Each element of an array can be accessed by an expression of the form:

```
arrayName[i]
```

`i` is called an index or subscript and may be any integer-valued expression. Typically a for loop is used to process array elements:

```
for (int i = 0; i < array.length; i++)
  // ... process array[i] ...
```

18

Examples:

```
for (int i = 0; i < studentScores.length; i++)
   sum += studentScores[i];

for (int i = 0; i < studentScores.length; i++)
   theScreen.println(studentNames[i] + " "
         + studentScores[i] + " " +
         + (studentScores[i] - theAverage) );

for (int i = 0; i < studentScores.length; i++) {
   theScreen.print((i + 1) + ": ");
   Keyboard.EatWhiteSpace();
   studentNames[i] = (theKeyboard.readLine());
   studentScores[i] = theKeyboard.readInt();
}
```

19

- Array *indexes may not get out of bounds*; i.e., less than 0 or greater than array.length – 1. If they do, a fatal **`ArrayIndexOutOfBoundsException`** is thrown.

```
for (int i = 0; i <= 4; i++)
  theScreen.println(array4[i]);   // Line 42
```

Sample run:

```
1.1
2.2
3.3
4.4
Exception in thread "main"
   java.lang.ArrayIndexOutOfBoundsException
   at testprogram.main(testprogram.java:42)
```

20

- *Arrays as parameters*: The *address/handle of an array argument is passed* to methods. Thus, as with class objects, modifying an array parameter will modify the corresponding array argument.

```
public double average(double [] anArray) {
  double sum = 0;
  for (int i = 0; i < anArray.length; i++)
    sum += anArray[i];
  return sum / anArray.length;
}
...
double [] a = {1.0, 2.0, 3.0};
theScreen.println("average: " + average(a) );
```

Note that **`average()`** can be used with a **`double`** *array of any size*. Not also the usefulness of the **`length`** *attribute to determine an array's size*.

21

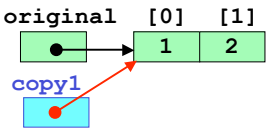
- *Arrays as return values*: Return type has the form ***`Type []`***; the method defines a local array of this same type and returns it. (Actually, it's address/handle is returned.)

```
public static int [] readArray() {
  // declare theScreen and theKeyboard
  theScreen.print("Enter the size of the array: ");
  int [] anArray = new int[theKeyboard.readInt()];

  theScreen.println("Enter the elements: ");
  for (int i = 0; i < anArray.length; i++)
    anArray[i] = theKeyboard.readInt();

  return anArray;
}
  . . .
int [] a = readArray();
```

22

- *Copying Arrays*: The same *aliasing problem* as for other objects.

```
int [] original = {1, 2};
int [] copy1;
copy1 = original;
```



Changing **`copy1`** changes **`original`**.
Use **`clone()`** (inherited from class **`Object`**) to produce a distinct copy.

```
int [] copy2;
copy2 = (int[])original.clone();
```

Need a deep-copy method for arrays of objects

23

- *Array Equality*: Same problem as for other objects:
  - **`anArray == anotherArray`** and **`anArray.equals(anotherArray)`** only check the handles of the two arrays.
  - Special equality-checking methods must be written if more that this is needed.
  - The **`String`** class has an **`equals()`** operator that checks string equality properly.

24

The Vector Class

```
Vector vec = new Vector();
int value;
theScreen.println("enter values: ");
for (;;){
  value = theKeyboard.readInt();
  if (value == -1) break;
  vec.addElement(new Integer(value));
}
```

+ `Vector`s can grow and shrink.

– `Vector` elements must be objects
  - types of numeric elements must be wrapper classes
  - many conversions between wrapper classes and primitive types

Also . . .


```
int sum = 0;
for (int i = 0; i < vec.size(); i++)
  sum += ((Integer)(vec.elementAt(i))).intValue();
theScreen.println("sum: " + sum);
```

– Parameters and return type of `Vector` methods are of type `Object` ⇒ typecasts must be used to convert to/from element type.

A better alternative: **`ArrayList`** (chap. 12)

Multidimensional Arrays

- **Problem:** To store student scores for more than one assignment.
- **One Solution:** Develop a program that can store scores for multiple students for multiple assignments, and, given the index of the student and the assignment number, display the appropriate score.

Preliminary Analysis

- The data structure required here could definitely use some sort of an array, but would be hard to fit into a one-dimensional array.
- Most languages provide multi-dimensional arrays for this sort of problem.

We can view a two-dimensional array as an array of arrays; that is, a one-dimensional array whose elements are one-dimensional arrays.

For example:

```
int [][] grades
  = { { 10, 17, 10, 18, 12 },
      { 20, 18, 14, 19, 19 },
      { 10, 14, 10, 14, 10 },
      { 15, 15, 15, 15, 15 },
      { 20, 19, 18, 17, 16 },
      { 20, 20, 19, 20, 20 } };
```

Storage is rowwise: Each internal array literal is a row of grades.

29

Accessing Array Elements

	[0]	[1]	[2]	[3]	[4]
[0]	10	17	10	18	12
[1]	20	18	14	19	19
[2]	10	14	10	14	11
[3]	15	15	15	15	15
[4]	20	19	18	17	16
[5]	20	20	19	20	20

`grades[0]` = row 0 of `grades`

`grades[1]` = row 1 of `grades`

`grades[1][3]` = 19

`grades[3][1]` = 15

30

```
for (int row=0; row < grades.length; row++) {
  for (int col=0; col < grades[row].length; col++)
    theScreen.print(grades[row][col] + "\t");
  theScreen.println();
}
```

Output:

```
10      17      10      18
12
20      18      14      19
19
10      14      10      14
10
15      15      15      15
15
20      19      18      17
```

31