Arrays and the ArrayList Class

Han-fen Hu

Tony Gaddis (2019) Starting Out with Java: From Control Structures through Data Structures, 4th Edition

UNLV

Outline

- Introduction to Arrays
- Two-Dimensional Arrays
- The **`ArrayList`** Class

Introduction to Arrays (1)

- Arrays allow us to create a collection of like values that are indexed.
  - An array can store any type of data but only one type of data at a time.
  - An array is a list of data elements.

```
// Declare a reference to an array that will hold
   integers.
// Create a new array that will hold 6 integers.
   int[] scores= new int[6];
```

0	0	0	0	0	0
index 0	index 1	index 2	index 3	index 4	index 5

Array element values are initialized to 0.

Array indexes always start at 0.

Introduction to Arrays (2)

- Arrays may be of any type, including strings and object.

```
float[] temperatures = new float[100];
char[] letters = new char[41];
long[] units = new long[50];
double[] prices= new double[1200];
String[] names = new String[4];
SavingsAccount[] accounts = new SavingsAccount[20];

final int ARRAY_SIZE = 6;
int[] numbers = new int[ARRAY_SIZE];
```

- Once created, an array size is fixed and cannot be changed.

UNLV

Introduction to Arrays (3)

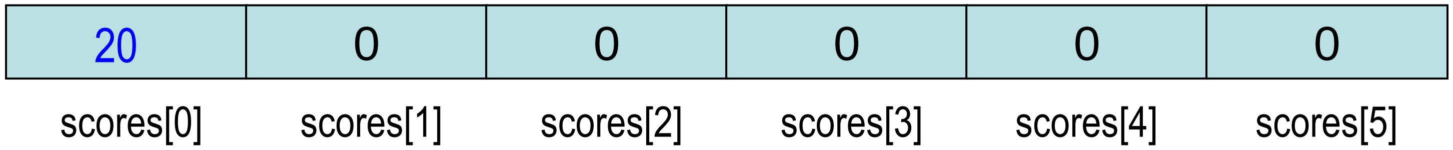
- You can let the user specify the size of an array:

```
// declare variables. Values not assigned.
 int numTests;
 int[] tests;

 // get user input for the number of tests
 Scanner keyboard = new Scanner(System.in);
 System.out.print("How many tests do you have? ");
 numTests = keyboard.nextInt();

 // use the input value to instantiate the array
 // of certain size
 tests = new int[numTests];
```

Introduction to Arrays (4)

- Each array element can be treated as a variable.
- They are simply accessed by the array name and a subscript.

```
scores[0] = 20;
```



- When relatively few items need to be initialized, an initialization list can be used to initialize the array.

```
int[]days = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30,
  31};

String[] names = { "Bill", "Carol", "Steven", "Jean" };
```

Introduction to Arrays (5)

- The **`length`** constant can be used in a loop to provide automatic bounding.

```
for(int i = 0; i < temperatures.length; i++){
  System.out.println("Temperature " + i ": "
                             + temperatures[i]);
}
```

- Array indexes always start at zero and continue to (array length - 1).

Introduction to Arrays (6)

- It is very easy to be off-by-one when accessing arrays.

```
// This code has an off-by-one error.
int[] numbers = new int[100];
for (int i = 1; i <= 100; i++)
   numbers[i] = 99;
```

  – Here, the equal sign allows the loop to continue on to index 100, where 99 is the last index in the array.

  – This code would throw an `ArrayIndexOutOfBoundsException`

Enhanced `for` Loop

- General format

```
for(datatype elementVariable : array)
   statement;
```

- Example

```
int[] numbers = {3, 6, 9};
for(int val : numbers){
  System.out.println("The next value is " + val);
}
```

- Simplified array processing (read only)
- Always goes through all elements

Arrays Operations (1)

- Arrays are objects.
  - Variables are used to reference the object in the memory
- An array reference can be assigned to another array of the same type.

```
// Create an array referenced by the numbers variable.
int[] numbers = new int[10];
// Reassign numbers to a new array.
numbers = new int[5];
```

- If the first (10-element) array no longer has a reference to it, it will be *garbage collected*.

Arrays Operations (2)

- You cannot copy an array by merely assigning one reference variable to another.
  - Both will referencing the same array. Updating the value in one will be reflected the other.
- You need to copy the individual elements of one array to another.

```
int[] firstArray = {5, 10, 15, 20, 25 };
int[] secondArray = new int[5];
for (int i = 0; i < firstArray.length; i++)
  secondArray[i] = firstArray[i];
```

- This code copies each element of **`firstArray`** to the corresponding element of **`secondArray`**.

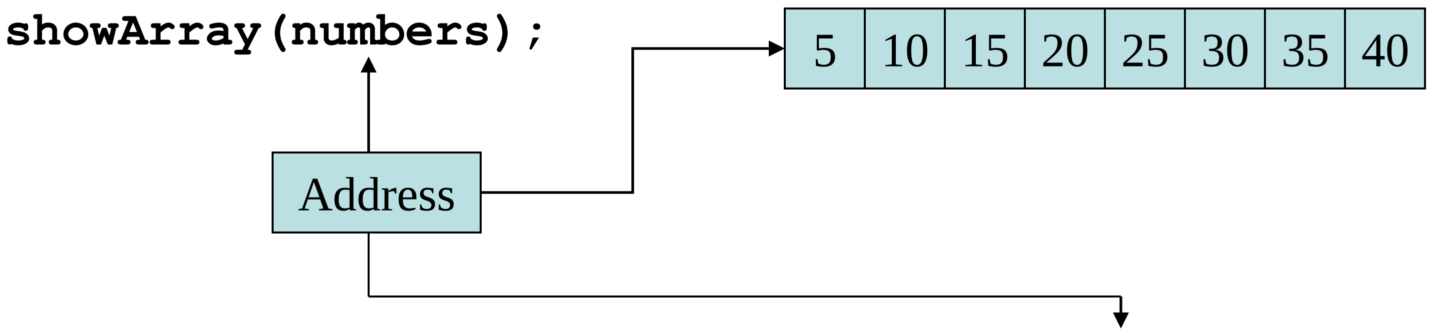
Reference to an Array

- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.

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

Passing Arrays as Arguments

- Arrays are objects.
- Their references can be passed to methods like any other object reference variable.



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

Lab (1)

- TestScoreConversion
  - In this example, the user would enter some scores and the program convert them to letter grades.

Returning an Array Reference

- A method can return a reference of an array.
- The return type of the method must be declared as an array of the right type.

```
public static double[] getArray(){
   double[] array = { 1.2, 2.3, 4.5, 6.7, 8.9 };
   return array;
}
```

  – In this example, the **`getArray()`** method is a public static method that returns an array of doubles.

Lab (2)

- Point.java
- BombGame.java
  - The BombGame program would read a series of five points, simulating the location of five cities. The program then would then ask the user to enter the location for the bomb, as well as the blast radius. The program would then show how many cities are affected.

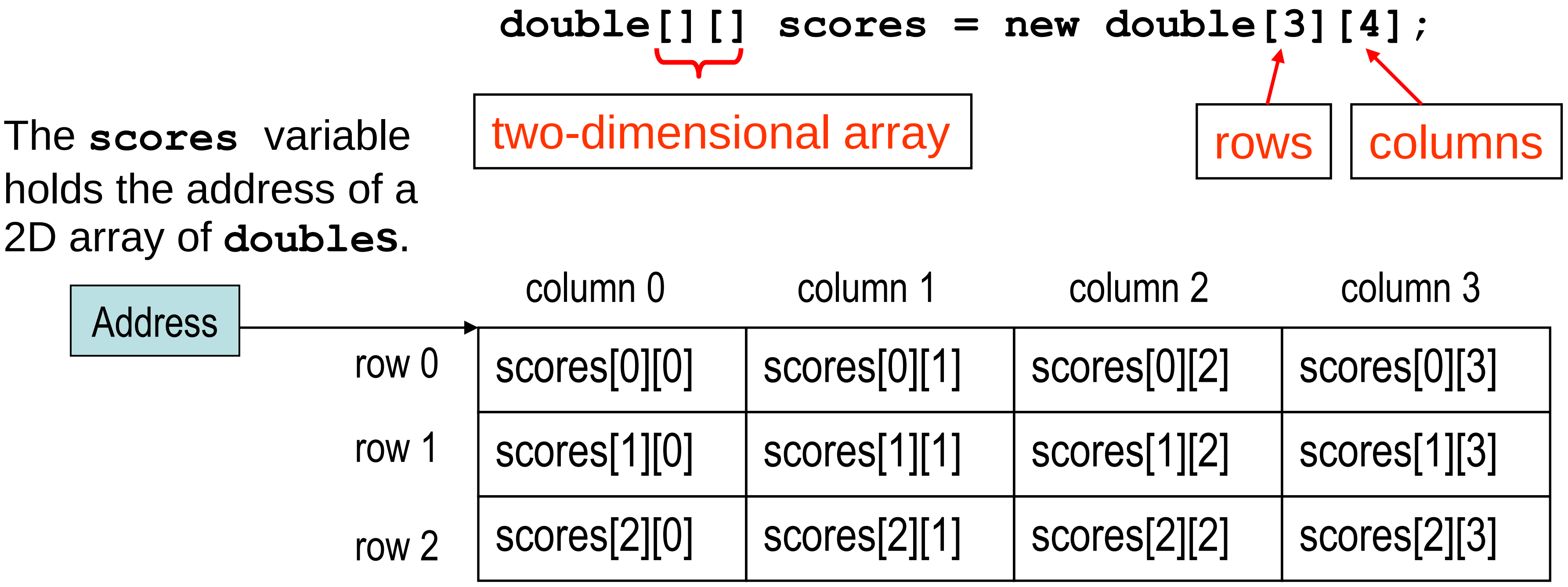
Two-Dimensional Arrays (1)

- A two-dimensional array is an array of arrays.
- It can be thought of as having rows and columns.
- Declaring a two-dimensional array requires two sets of brackets and two size declarators
  - The first one is for the number of rows
  - The second one is for the number of columns.

```
double[][] scores = new double[3][4];
```

	column 0	column 1	column 2	column 3
row 0				
row 1				
row 2				

Two-Dimensional Array Elements (2)



double[][] scores = new double[3][4];
two-dimensional array
rows
columns
The scores variable holds the address of a 2D array of doubles.
Address

	column 0	column 1	column 2	column 3
row 0	scores[0][0]	scores[0][1]	scores[0][2]	scores[0][3]
row 1	scores[1][0]	scores[1][1]	scores[1][2]	scores[1][3]
row 2	scores[2][0]	scores[2][1]	scores[2][2]	scores[2][3]

- When processing the data in a two-dimensional array, each element has two subscripts:
  - one for its row and
  - another for its column.

Two-Dimensional Array Elements (3)

- Programs that process two-dimensional arrays can do so with nested loops.
- To fill the scores array:

Number of rows, not the largest subscript

```
for (int row = 0; row < 3; row++){
  for (int col = 0; col < 4; col++){
    System.out.print("Enter a score: ");
    scores[row][col] = keyboard.nextDouble();
  }
}
```

Number of columns, not the largest subscript

Two-Dimensional Array Elements (4)

- Initializing a two-dimensional array requires enclosing each row's initialization list in its own set of braces.

```
int[][] numbers = { {1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
```

- Java automatically creates the array and fills its elements with the initialization values.
  - row 0 {1, 2, 3}
  - row 1 {4, 5, 6}
  - row 2 {7, 8, 9}
- Declares an array with three rows and three columns.

Two-Dimensional Array Elements (5)

- To access the **`length`** fields of the array:

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

for (int row = 0; row < numbers.length; row++) {
  for (int col = 0; col < numbers[row].length; col++)
    System.out.println(numbers[row][col]);
}
```

Number of rows

Number of columns in this row.

Ragged Arrays

- When the rows of a two-dimensional array are of different lengths, the array is known as a *ragged array*.
- You can create a ragged array by creating a two-dimensional array with a specific number of rows, but no columns.

```
int [][] ragged = new int [4][];
```

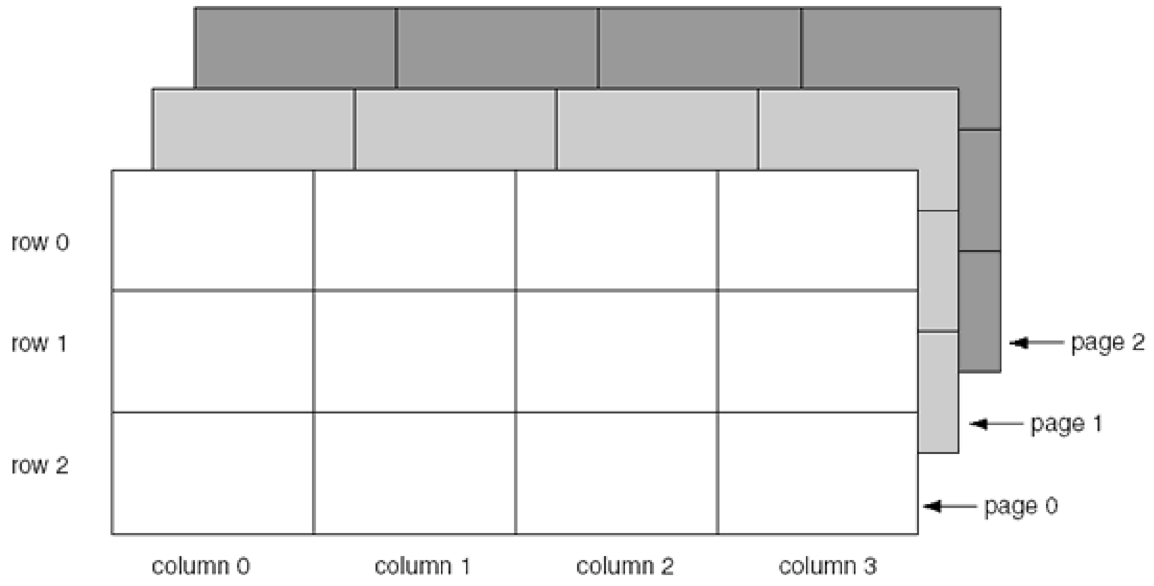
- Then create the individual rows.

```
ragged[0] = new int [3];
ragged[1] = new int [5];
ragged[2] = new int [6];
ragged[3] = new int [8];
```

UNLV

More Than Two Dimensions

- Java does not limit the number of dimensions that an array may be.
- More than three dimensions is hard to visualize, but can be useful in some programming problems.



`ArrayList` Class (1)

- Similar to an array, an **`ArrayList`** allows object storage
- Unlike an array, an **`ArrayList`** object:
  - Automatically expands when a new item is added
  - Automatically shrinks when items are removed
- Requires:

```
import java.util.ArrayList;
```

`ArrayList` Class (2)

- An **`ArrayList`** has a capacity, which is the number of items it can hold without increasing its size.
- The default capacity of an **`ArrayList`** is 10 items.
- To designate a different capacity, use a parameterized constructor:

```
ArrayList<String> list = new ArrayList<String>(100);
```

ArrayList Class (3)

- Creating an **ArrayList**

```
ArrayList<String> nameList = new ArrayList<String>();
```

  - Notice the word **String** written inside angled brackets <>

- To populate the **ArrayList**, use the **add()** method:

```
    nameList.add("James");
    nameList.add("Catherine");
```

- The **ArrayList** class's **remove()** method removes designated item from the **ArrayList**

```
    nameList.remove(1);
```

  This statement removes the second item.

`ArrayList` Class (4)

- To get the current size, call the **`size()`** method

```
nameList.size();   // returns 2
```

- To access items in an **`ArrayList`**, use the **`get()`** method

```
nameList.get(1);
```

In this statement 1 is the index of the item to get.

- The **`ArrayList`** class's **`toString()`** method returns a string representing all items in the **`ArrayList`**

```
System.out.println(nameList);
```

This statement yields :

```
[ James, Catherine ]
```

`ArrayList` Class (5)

- To insert items at a location of choice, use the **`add()`** method with two arguments:

  `nameList.add(1, "Mary");`

  This statement inserts the **`String`** "Mary" at index 1

- To replace an existing item, use the **`set()`** method:

  `nameList.set(1, "Becky");`

  This statement replaces “Mary” with “Becky”

UNLV

Lab (3)

- FriendList.java
  - The program would ask the user to enter a name, and add it to the ArrayList.

Using an **`ArrayList`**

- You can store any type of *object* in an **`ArrayList`**

```
ArrayList<BankAccount> accountList =
                 new ArrayList<BankAccount>();
```

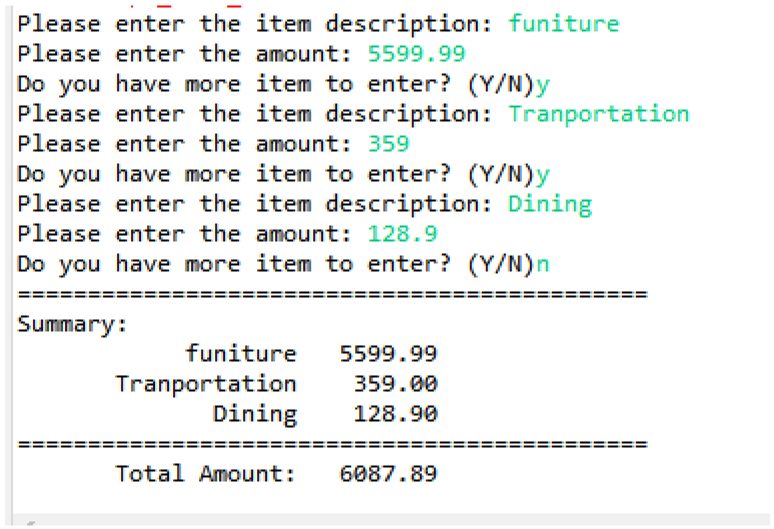
This creates an **`ArrayList`** that can hold **`BankAccount`** objects.

Lab (4)

- Revise BombGame.java
  - Now we would revise it to use ArrayList, so that it would allow reading unlimited number of cities.

Class Exercise

- Expense Summary
  - Please write a program that allows user to enter expenses and amounts. The program then shows a summary and the total amount.
  - You can use arrays or ArrayList class to complete this exercise.
  - To declare ArrayList of double values, you need to use `ArrayList<Double> amounts = new ArrayList<Double>();`



```
Please enter the item description: funiture
Please enter the amount: 5599.99
Do you have more item to enter? (Y/N)y
Please enter the item description: Tranportation
Please enter the amount: 359
Do you have more item to enter? (Y/N)y
Please enter the item description: Dining
Please enter the amount: 128.9
Do you have more item to enter? (Y/N)n
=============================================
Summary:
            funiture   5599.99
       Tranportation    359.00
              Dining    128.90
=============================================
       Total Amount:   6087.89
```