NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#8: Arrays

Objectives

- Motivation behind using arrays
- Introducing arrays as data structures
- Learning how to declare and use one dimensional arrays
- Learning how to declare and use two dimensional arrays

Motivation

Let s take a moment to consider this scenario

Often you will have to store a large number of values during the execution of a program. Suppose, for instance, that you need to read one hundred numbers, compute their average, and find out how many numbers are above the average.

Your program first reads the numbers and computes their average, and then compares each number with the average to determine whether it is above the average. The numbers must all be stored in variables in order to accomplish this task. You have to declare one hundred variables and repeatedly write almost identical code one hundred times. From the standpoint of practicality, it is impossible to write a program this way.

So, how do you solve this problem?

In class discussion

Introduction to Arrays

- An **array** is a collection of individual data values with two distinguishing characteristics:
  - An array is ordered (not sorted). You must be able to count off the values: here is the first, here is the second, and so on.
  - An array is homogeneous. Every value in the array must have the same type.

Introduction to Arrays

- The individual values in an array are called elements. The type of those ***elements*** (which must be the same (of the same type) because arrays are homogeneous) is called the element type. The number of elements is called the length of the array.

- Each element is identified by its position number in the array, which is called its index. In Java, index numbers always begin with 0 and therefore extends up to one less than the length of the array.

Declaring an Array Variable

- As with any other variable, array variables must be declared before you use them. In Java, the most common syntax for declaring an array variable looks like this:

```
type[] name = new type[n];
```

where type is the element type, name is the array name, and n is an integer expression indicating the number of elements.

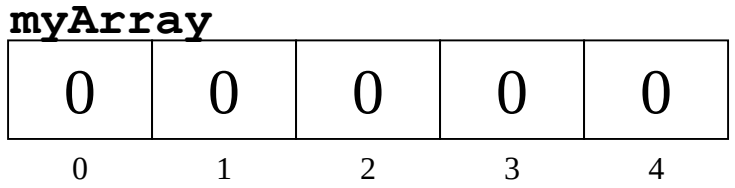
- This declaration syntax combines two operations. The part of the line to the left of the equal sign declares the variable; the part to the right creates an array value with the specified number of elements and then assigns it to the array variable.
- Even though the two operations are distinct, it will help you avoid errors if you make a habit of initializing your arrays when you declare them.

Example of one dimensional array

- The following declaration creates an array called *myArray* consisting of 5 values of type int:

```
int[] myArray = new int[5];
```

- This easiest way to visualize arrays is to think of them as a linear collection of boxes, each of which is marked with its index number. You might therefore diagram the *myArray* variable by drawing something like this:



- Java automatically initializes each element of a newly created array to its default value, which is zero for numeric types, false for values of type boolean, and null for objects.

Accessing elements in array

- Given an array such as the **`myArray`** variable you can get the value of any element by writing the index of that element in brackets after the array name. This operation is called **selection**.
- You can, for example, select the initial element by writing

```
myArray[0]
```

- The result of a selection operation is essentially a variable. In particular, you can assign it a new value. The following statement changes the value of the first element to 4:

```
myArray[0] = 4;
```

Cycling through an array

- One of the most useful things about array selection is that the index does not have to be a constant. In many cases, it is useful to have the index be the control variable of a **for** loop.

- The standard **for** loop pattern that cycles through each of the array elements in turn looks like this, the second loop initializes all elements of array to 0:

```
for (int i = 0; i < array.length; i++) {
   Operations involving the ith element of the array
}
```

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

- array.**length** field returns the number of elements.

Notes on one Dimensional Arrays

Declaring & Creating Arrays

Syntax for declaration of a one dimensional array:

elementType[ ] arrayName;

This does not allocate memory for the array

Syntax for creation of a one dimensional array:

arrayName = new elementType[numberOfElements];

This allocates memory to store the specified number of variables of type elementType.

Example:

```
int [] assignmentScores;
assignmentScores = new int[15];
```

Declaring & Creating Arrays

Creating and declaring an array in one step

elementType [] arrayName = {elementOne, elementTwo, elementThree…}

int [] assignmentScores = {9, 10, 10, 0, 8, 7, 10, 10, 9, 9};

- If an array has been declared, but not initialized, then it is filled with the default value of the variable type.

- The length of an array cannot be changed once declared. The size can be determined using arrayName.length.

Accessing Individual Elements

- Use the following syntax to access the elements of an array:

arrayName[index]

Where index is 0 <= index < numberOfElements

- Example:

```
System.out.printf( “element at position %n is %f  \n”, i, array[i]);
array[17] = 26.116;
```

Processing Arrays

- There are a number of different operations that allow you to process arrays once they have been created.
- **Initializing** arrays to a random, predefined, or user-defined value:

```
double [] numbers = new double [50];
for (int i = 0; i < numbers.length; i++){
    ...
    numbers[i] = value;
}
```

Processing Arrays

- Adding all of the elements of an array:

```
double [] numbers = new double [50];
...  //initialize the values in the array
double sum = 0.0;
for (int i = 0; i < numbers.length; i++){
    sum = sum + numbers[i];
}
```

Processing Arrays

- Finding the smallest value in an array of numbers

```
double [] numbers = new double [50];
...  //initialize the values in the array
double minValue = numbers[0];
int minIndex = 0;
for (int i = 1; i < numbers.length; i++){
    if (numbers[i] < minValue )
    {
        minValue = numbers[i];
        minIndex = i;
    }
}
```

Processing Arrays

- Finding the largest value in an array of numbers

```
double [] numbers = new double [50];
...  //initialize the values in the array
double maxValue = numbers[0];
int maxIndex = 0;
for (int i = 1; i < numbers.length; i++){
    if (numbers[i] >= maxValue )
    {
        maxValue = numbers[i];
        maxIndex = i;
    }
}
```

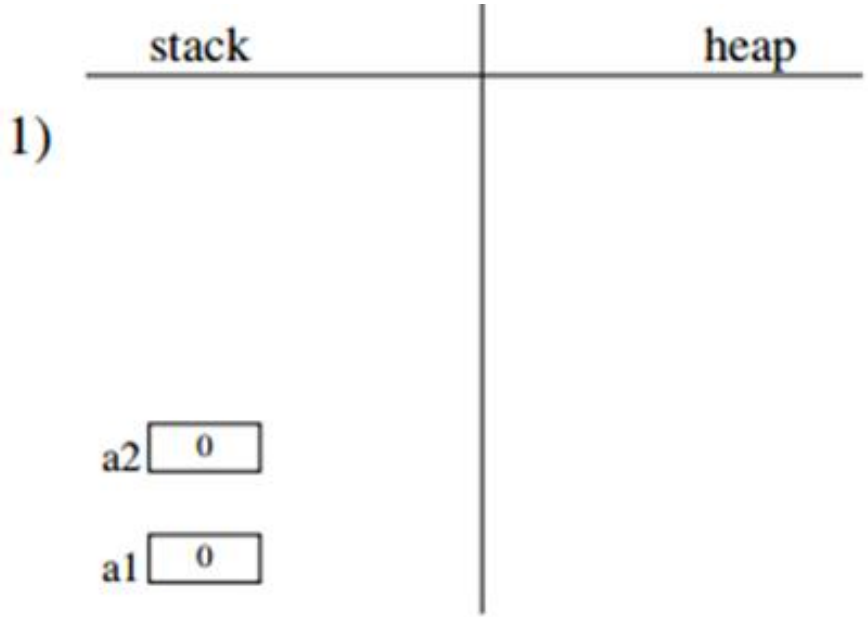
Copying Arrays

- Arrays in Java are implemented as objects, which means that they are stored in the heap *(the heap is the portion of memory where dynamically allocated memory resides)* **The value stored in an array variable is simply a reference to the actual array.**

- The memory allocated for an array is located on the heap of the program's memory. The name of the array itself is a reference variable stored on the stack;

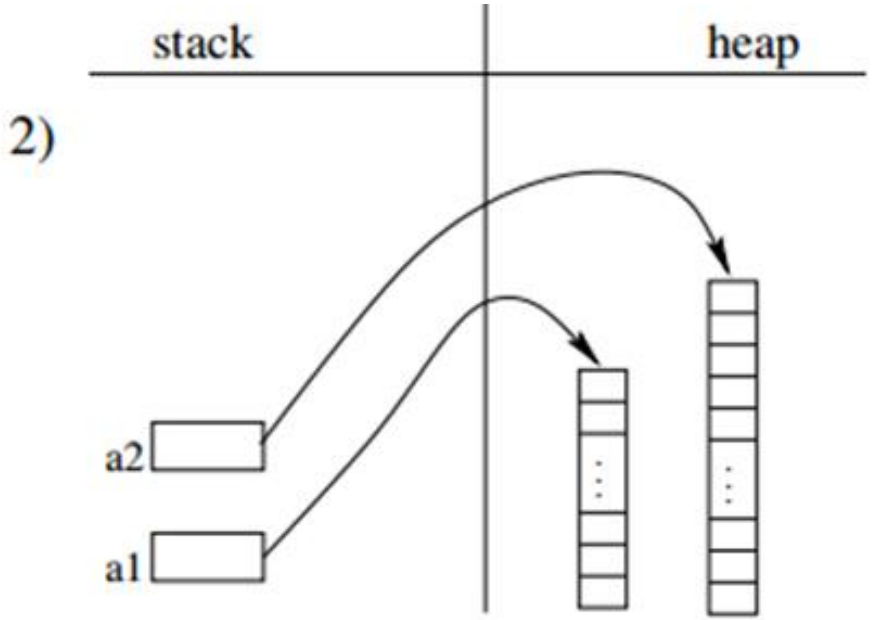
- The reference variable can contain either null (if the array has not been created) or the address of the memory on the heap.

Copying Arrays



When arrays are declared, no memory is allocated on the heap.

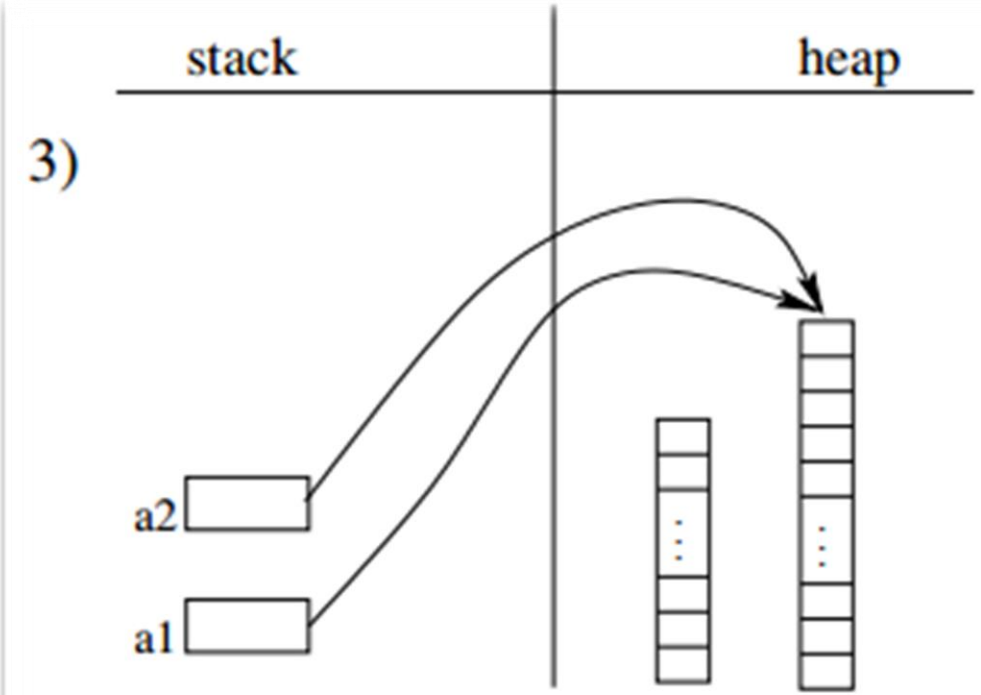
```
int [] a1;
int [] a2;
```



When arrays are created, the name of the variable contains a reference to the array location on the heap.

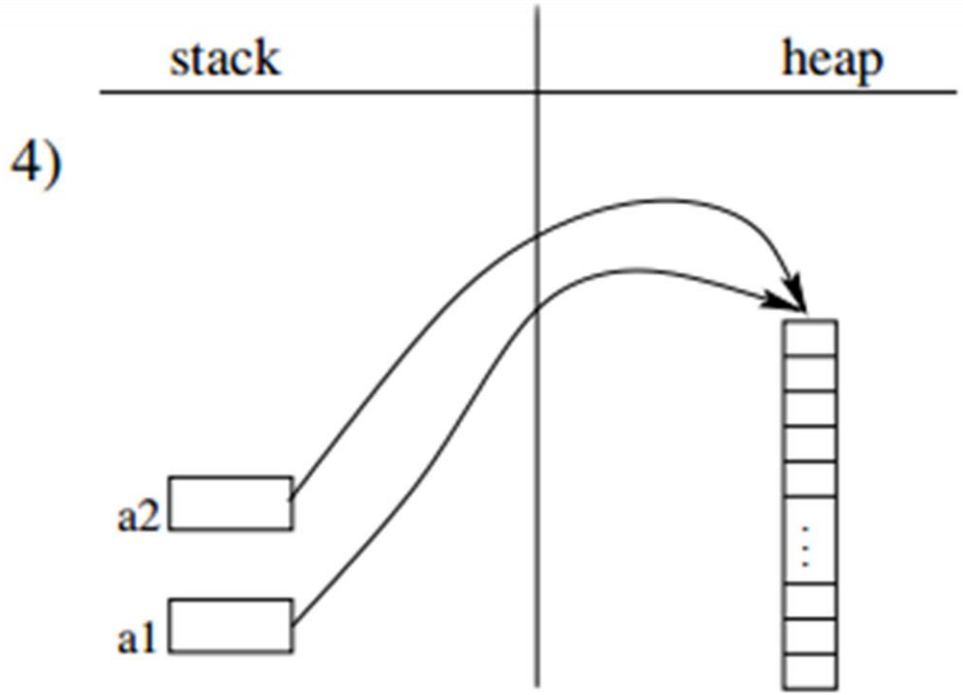
```
a1 = new int [15];
a2 = new int [27];
```

Copying Arrays



If one array name is assigned another memory name, only the reference stored changes. This is not a way of copying one array into another:

```
a1 = a2;
```



There is no longer a way of accessing the elements of the first array, so the Java garbage collector eventually removes it from the heap.

Copying Arrays

- If you wish to copy the content of an array, you will need to copy each element one at a time. The size of the new array must be the same as the original.

- Example:

```
for (int i = 0; i < a1.length; i++){
    a1[i] = a2[i];
}
```

Using Arrays in Methods

- Arrays, like variables, can be used as parameters and returned as values from methods.
- **When you pass an array as a parameter to a method or return a method as a result, only the reference to the array is actually passed between the methods.**
- The effect of Java's strategy for representing arrays internally is that the elements of an array are effectively shared between the caller and callee. If a method changes an element of an array passed as a parameter, **that change will persist after the method returns.**
- Passing Arrays into Methods:
  - When an array is passed into a method, the reference variable is used. **This means that the method can modify the contents of an array.**
  - Syntax:

```
modifiers returnType methodName (arrayType [] arrayName ){
    method body
}
```

Using Arrays in Methods

- Example: What will be the result of the following code?

```
1  public class TestArrayParameters {
2
3      public static void main (String[] args ) {
4          int i; //loop counter variable
5          double [] myNums = {1.1, 2.2, 3.3, 4.4, 5.5};
6
7          for ( i = 0; i < myNums.length; i++)
8              System.out.print( myNums[i] + ", ");
9
10         multiplyBy( myNums    , 2.0);
11
12         for ( i = 0; i < myNums.length; i++)
13             System.out.print( myNums[i] + ", ");
14     }
15
16     public static void multiplyBy(double[] numbers, double multiplier)
17     {
18         for (int i = 0; i < numbers.length; i++) {
19             numbers[i] *= multiplier;
20         }
21     }
22 }
```

Using Arrays in Methods

- Returning Arrays from Methods:
  - When a method returns an array, the reference variable is returned.

  Syntax:

```
modifiers returnType [] methodName (parameters) {
    method body
}
```

Using Arrays in Methods

- Example: What will be the result of the following code?

```
1  public class TestReturnedArray {
2
3      public static void main (String[] args ) {
4          int [] myNumbers = createRandomIntArray( 15 );
5          if (myNumbers.length != 15)
6              System.err.println("Something went wrong!");
7          for (int i = 0; i < myNumbers.length; i++)
8              System.out.println( myNumbers[i] );
9      }
10
11     public static int [] createRandomIntArray( int size )
12     {
13         int [] randomNumbers = new int [size];
14
15         for (int i = 0; i < randomNumbers.length; i++) {
16             randomNumbers[i] = (int) ( Math.rand() *
17                 (Integer.MAX_VALUE - Integer.MIN_VALUE + 1) )
18                 + Integer.MIN_VALUE;
19         }
20     }
21 }
```

Searching through Arrays

- We will focus on two search techniques: Linear Search and Binary Search.

- Linear Search:

  - This algorithm searches an array for a specific item by starting at the first item in the array and compares each element in the array to a key variable until the item is found or the end of the array is reached.

  - If the variable is found, the return value is the location within the array. If not, the return value indicates a failure, such as -1.

Searching through Arrays

- Linear Search Example:

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

- Questions:
  - How many locations will be examined in the worst case scenario?
  - How many locations will be examined in the best case scenario?
  - How many locations will be examined on average?

Searching through Arrays

- Binary Search
  - This algorithm searches an array for a specific key, but it discards half of the remaining list at each step.
  - Take note that this algorithm assumes the array has already been sorted.
  - Because the array should already have been sorted, this search algorithm can be completed in fewer steps.

Binary Search Algorithm

```
public static int binarySearch ( double [] list, double key )
{
    int low = 0;
    int hight = 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;
}
```

Searching through Arrays

Questions:

- How many locations will be examined in the worst case scenario?
- How many locations will be examined in the best case scenario?
- How many locations will be examined on average?
- How many operations will it take to sort the array first?

Sorting Arrays

- We will focus on two sorting techniques: Selection Sort and Insertion Sort
- Selection Sort:
  - This is a sorting algorithm that starts by finding the smallest item in the array and swaps it with the first element of the list. Next, it finds the second smallest element and swaps it with the second item in the array. This process continues until the array has been fully sorted from smallest to largest.

Selection Sort

```
public static void selectionSort(double[] list) {
    for (int i = 0; i < list.length - 1; 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;
        }
    }
}
```

Sorting Arrays

- Insertion Sort:
  - Assume that you want to sort the contents of an array from smallest to largest. This algorithm repeatedly inserts elements into a sorted sub-list until the whole array is sorted.

Insertion Sort Example

```
public static void insertionSort(double[] list) {
    for (int i = 1; i < list.length; i++) {
        /** insert list[i] into a sorted sublist list[0..i-1] so that
        list[0..i] is sorted.  */
        double currentElement = list[i];
        int k;
        for (k = i - 1; k >= 0 && list[k] > currentElement; k--) {
            list[k + 1] = list[k];
        }
        // Insert the current element into list[k + 1]
        list[k + 1] = currentElement;
    }
}
```

Insertion Sort Example

- Question:
  - How many operations will it take to sort an array?

Notes on Two-Dimensional Arrays

Introduction to Multidimensional Arrays

- Because the elements of an array can be of any Java type, those elements can themselves be arrays. Arrays of arrays are called multidimensional arrays.
- In Java, you can create a multidimensional array by using multiple brackets in both the type and the initialization parts of the declaration. For example, you can create array space for a 3 x 3 tic-tac-toe board using the following declaration:

```
char[][] board = new char[3][3];
```

Introduction to Multidimensional Arrays

- Because the elements of an array can be of any Java type, those elements can themselves be arrays. Arrays of arrays are called multidimensional arrays.
- In Java, you can create a multidimensional array by using multiple brackets in both the type and the initialization parts of the declaration. For example, you can create array space for a 3 x 3 tic-tac-toe board using the following declaration:

```
char[][] board = new char[3][3];
```

Introduction to Multidimensional Arrays

- This declaration creates a two-dimensional array of characters that is organized like this:

```
char[][] board = new char[3][3];
```

board[0][0]	board[0][1]	board[0][2]
board[1][0]	board[1][1]	board[1][2]
board[2][0]	board[2][1]	board[2][2]

Declaring & Creating Arrays Two-Dimensional Arrays

- You may think of a two-dimensional array as “an array of arrays” or, more simply, a construct containing rows and column

- Declaration Syntax:

elementType [] [] arrayName;

Declaring & Creating Arrays Two-Dimensional Arrays

- Creation syntax:
  - Create each row of the same size:

arrayName = new elementTy[e[numberOfRows] [numberOfColumns];

  - Create an array with different sizes

arrayName = new elementTy[e[numberOfRows] [];

```
for (int i = 0; i <arrayName.length; i++) {
    arrayName[i] = new elementType [numberofColsInRowI];
}
```

Accessing Individual Elements: Two-Dimensional Arrays

- Using the row and column index, instead of just the row index, one can access each element of an array using the following syntax:

arrayName[rowIndex] [columnIndex];

Processing Two-Dimensional Arrays

- Initializing arrays to a random, predefined, or user-defined value:

```
double [][] numbers = new double [50][150];
for (int row = 0; row < numbers.length; row++){
    for (int col = 0; col < numbers[row].length; col++){
    ...
    numbers[row][col] = value;
}
```

Processing Two-Dimensional Arrays

- Printing values on a diagonal of a square two-dimensional array:

```
double [][] numbers = new double [50][50];
...  //initialize the values in the array
for (int row = 0; row < numbers.length; row++){
    //make sure the array is square
    if (numbers.length != numbers[row].length )
        System.err.println("Error:  array is not square!");
    System.out.println( numbers[row][row]);
}
```

Processing Two-Dimensional Arrays

- Questions:
  - How would you find smallest element in each row / each column?
  - How would you determine which row/column has the largest element in the array?
  - How would you sort each row/column of the array?

Important Points to Focus on (discussed in class)

Revisit the previous chapter's notes, this chapter's notes and the reading about arrays and methods from the book's chapter

- Passing arguments by reference and passing argument by value
- Passing an array to a method (does the content change?)
- Global variable and local variables
- Example: swapping two variables using a Method in Java (Declaring the variables as global or local in the main and passing them to the method swap)
- Method overloading

NYU

Courant Institute of Mathematical Sciences
Department of Computer Science
CS101 Introduction to Computer Science

Anasse Bari, Ph.D.

Chapter#8: Arrays