Arrays

Introduction to Programming and Computational Problem Solving: Accelerated Pace

CSE 11

Lecture 8
Announcements

• Assignment 3 is due today, 11:59 PM
  – Upgrade beginning Apr 27, 12:01 AM
• Assignment 4 will be released today
  – Due May 1, 11:59 PM
Arrays

- An array is a data structure that represents a collection of the same types of data.
Declaring array variables

datatype[] arrayRefVar;
• For example
double[] myList;

datatype arrayRefVar[];
• For example
double myList[];

If a variable does not contain a reference to an array, the value of the variable is null

This style is allowed, but not preferred
Creating arrays

arrayRefVar = new datatype[arraySize];

• For example
  
  myList = new double[10];

  • myList[0] references the first element in the array
  • myList[9] references the last element in the array
Declaring and creating in one step

datatype[] arrayRefVar = new datatype[arraySize];

• For example
  
double[] myList = new double[10];
The length of an array

• Once an array is created, its size is fixed (i.e., it cannot be changed)
• You can find its size using `arrayRefVar.length`
• For example,
  
  ```java
  double[] myList = new double[10];
  myList.length returns 10
  ```
Default values

• When an array is created, its elements are assigned the default value of:
  0 for the numeric primitive data types
  \u0000 for char type
  false for boolean type
Indexed variables

• The array elements are accessed through the index

• The array indices are **0-based**
  – From 0 to arrayRefVar.length - 1

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

  arrayRefVar[index];
Using indexed variables

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

• For example

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

• Declaring, creating, and initializing in one step
  
  ```java
  double[] myList = {1.9, 2.9, 3.4, 3.5};
  ```

• This shorthand syntax must be in one statement
  – The above statement is equivalent to the following statements
  
  ```java
  double[] myList = new double[4];
  myList[0] = 1.9;
  myList[1] = 2.9;
  myList[2] = 3.4;
  myList[3] = 3.5;
  ```
Initializing arrays

- Initializing arrays with input values
  ```java
  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
  ```java
  for (int i = 0; i < myList.length; i++) {
      myList[i] = Math.random() * 100;
  }
  ```
Processing arrays

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

• Finding the element with the maximum value
  
  ```java
double max = myList[0];
  for (int i = 1; i < myList.length; i++) {
    if (myList[i] > max) max = myList[i];
  }
  ```
Printing arrays

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

• Traverse the complete array **sequentially** without using an index variable
  ```java
  for (elementType value : arrayRefVar) {
    // Process the value
  }
  ```
  
• For example
  ```java
  for (double value : myList)
    System.out.println(value);
  ```

• You still must use an index variable if you wish to traverse the array in a different order or change the elements in the array
Copying arrays

• The assignment statement does not copy the contents, it only copies the reference value

```java
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 list2
```
Copying arrays

• To copy contents of one array to another, you must copy the array’s individual elements to the other array

• Using a loop

```java
int[] srcArray = {2, 3, 1, 5, 10};
int[] destArray = new int[srcArray.length];
for (int i = 0; i < destArray.length; i++)
    destArray[i] = srcArray[i];
```
Copying arrays

• Using the System.arraycopy method
  – Java 8 API documentation
    • https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#arraycopy-java.lang.Object-int-java.lang.Object-int-int-
  – Java 11 API documentation

System.arraycopy(srcArray, srcPos, destArray, destPos, length);

– For example:

```java
int[] srcArray = {2, 3, 1, 5, 10};
int[] destArray = new int[srcArray.length];
System.arraycopy(srcArray, 0, destArray, 0, destArray.length);
```
Passing arrays to methods

• When passing an array to a method, the **reference** of the array is passed to the method

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

Invoke the method, example 1:
```java
int[] list = {3, 1, 2, 6, 4, 2};
printArray(list);
```

Invoke the method, example 2:
```java
printArray(new int[]{3, 1, 2, 6, 4, 2});
```
Anonymous array

• The statement
  
  ```java
  printArray(new int[]{3, 1, 2, 6, 4, 2});
  ```
  
creates an array using the syntax
  
  ```java
  new dataType[]{literal0, literal1, ..., literalN};
  ```
  
• There is no explicit reference variable for the array (i.e., it is unnamed)

• Such an array is called an *anonymous array*
Pass by value

• Remember, Java uses **pass by value** to pass arguments to a method

• For a parameter of a primitive type, 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 **reference value** is passed
  – Any changes to the array that occur inside the method body **does** affect the original array that was passed as the argument
public class Test {
    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]
}
Pass by value

• 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.

![Diagram showing pass by value and reference to array in stack and heap.]

Arrays are stored in a heap.
Heap

- The Java Virtual Machine (JVM) stores the array in an area of memory called the *heap*, which is used for dynamic memory allocation.
Returning an array from a method

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

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

- The `java.util.Arrays` class contains useful methods for common array operations
  
  - Sorting arrays
    - For example, `java.util.Arrays.sort`
  
  - Searching arrays
    - For example, `java.util.Arrays.binarySearch` (a sorted in ascending order array)
  
  - Check whether two arrays are strictly equal
    - `java.util.Arrays.equals`
  
  - Fill all or part of an array
    - `java.util.Arrays.fill`
  
  - Return a string that represents all elements in an array
    - `java.util.Arrays.toString`
Command-line parameters

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

java TestMain arg0 arg1 arg2 ... argn

• In the main method, get the arguments from args[0], args[1], ..., args[n], which corresponds to arg0, arg1, ..., argn in the command line
Two-dimensional arrays

// Declare array reference variable
dataType[][][] refVar; // preferred
dataType refVar[][][];

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

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

// Alternative syntax
dataType refVar[][][] = new dataType[arrayRows][arrayCols];

If a variable does not contain a reference to an array, the value of the variable is null.
Two-dimensional arrays

• You can also use an array initializer to declare, create, and initialize a two-dimensional array

• For example

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

```java
int[][] array = new int[4][3];
array[0][0] = 1; array[0][1] = 2; array[0][2] = 3;
array[1][0] = 4; array[1][1] = 5; array[1][2] = 6;
array[2][0] = 7; array[2][1] = 8; array[2][2] = 9;
array[3][0] = 10; array[3][1] = 11; array[3][2] = 12;
```

A two-dimensional array is an **array of arrays**
Two-dimensional arrays

A two-dimensional array is an **array of arrays**

```java
int[][] array = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9},
    {10, 11, 12}
};
```
Lengths of two-dimensional arrays

- A two-dimensional array is an **array of arrays**
  
  ```java
  int[][] x = new int[3][4];
  ```

  ![Diagram of a 2D array]

- Remember, last array is `x[x.length - 1]`
Ragged arrays

• Each row in a two-dimensional array is itself an array
• The rows can have different lengths
• If so, then the array is called a *ragged array*

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

triangleArray.length is 5
triangleArray[0].length is 5
triangleArray[1].length is 4
triangleArray[2].length is 3
triangleArray[3].length is 2
triangleArray[4].length is 1
Initializing two-dimensional arrays

• Initializing arrays with input values

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

• Initializing arrays with random values

```java
for (int row = 0; row < matrix.length; row++) {
    for (int column = 0; column < matrix[row].length; column++) {
        matrix[row][column] = (int)(Math.random() * 100);
    }
}
Two-dimensional arrays

• Nested for loops are often used to process a two-dimensional array
• When passing a two-dimensional array to a method, the reference of the array is passed to the method
  – Just like methods and one-dimensional arrays
  – Any changes to the array that occur inside the method body will affect the original array that was passed as the argument
Higher dimensional 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 two-dimensional arrays can be generalized to declare $n$-dimensional array variables and create $n$-dimensional arrays for $n \geq 3$
Three-dimensional arrays

- A three-dimensional array is an array of two-dimensional arrays

```java
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}}
};
```

Which student

Which exam

Multiple-choice or essay
Next Lecture

• Objects and classes