Lecture Feb 21 - Multi Dim. Arrays

Bentley James Oakes

February 20, 2018

Midterm

- Midterm on **March 13th, 18:00 to 21:00**
- On Monday I'll talk about which material will be on the midterm
  - This class is included
- Monday, March 12th I'll go through some example questions

Format:

- True/False
- Short Answer
- Long Answer
  - Writing code by hand

This Lecture

Section 1

Changing Elements in an Array

Changing Elements in an Array

```
public static void main(String[] args){
    int[] a = {7, 4, 5, 2, 6};
    doubleArray(a);

    System.out.println(Arrays.toString(a));
    //prints [14, 8, 10, 4, 12]
}

public static void doubleArray(int[] arr){
    for (int i=0; i < arr.length; i++){
        arr[i] = arr[i] * 2;
    }
}
```

- Method to double each element in the array
- Note that this method is `void`, but the elements are still modified
- Void just means the method does not return a value

Creating a New Array

```
public static void main(String[] args){
    int[] a = {1, 6, 3, 7};
    System.out.println(a);
    //address stored in array [I@72510787

    System.out.println(Arrays.toString(a));
    //aray elements [1, 6, 3, 7]

    create(a);
}

public static void create(int[] a){
    //create a new array
    int[] b = new int[2];

    System.out.println(b);
    //address of the new array [I@194f6c50

    System.out.println(Arrays.toString(b));
    //default values stored in array[0, 0]

    //changes to elements in b do not affect array a
}
```

Creating a New Array

```
public static void main(String[] args){
    int[] a = {1, 6, 3, 7};
    System.out.println(a);
    //address stored in array [I@188c083d

    System.out.println(Arrays.toString(a));
    //aray elements [1, 6, 3, 7]

    create(a);
}

public static void create(int[] a){
    //create a new array
    a = new int[2];

    System.out.println(a);
    //address of the new array [I@69d3b600

    System.out.println(Arrays.toString(a));
    //default values stored in array[0, 0]

    //this variable a points to a different location
}
```

Copying Arrays

```
public static void main(String[] args){
    int[] a = {7, 4, 5, 2, 6};
    int[] b = copyArray(a);

    System.out.println(Arrays.toString(b));
    //prints [7, 4, 5, 2, 6]

    b[0] = 100;
    System.out.println(Arrays.toString(b));
    //prints [100, 4, 5, 2, 6]
}

public static int[] copyArray(int[] arr){
    int[] result = new int[arr.length];
    for (int i=0; i < arr.length; i++){
        result[i] = arr[i];
    }
    return result;
}
```

Section 2

Revisiting Strings

Revisiting Strings

- Let's go back to `Strings`
- `Strings` are a reference type, but they are special
- The data pointed to by a `String` variable can't be changed after it has been created
    - `Strings` are **immutable**
    - Immutable means the data can't be changed

Immutable Strings

Behind the scenes, it looks more like this:

```
String s = "apples";
String temp = s + " and bananas";
s = temp;
```

- New `String` data is created with every concatenation
- The original variable then automatically points to the new data

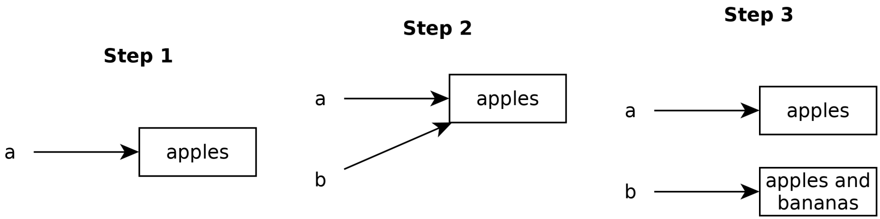
String Aliasing

```
String a = "apples"; //Step 1:

String b = a; //Step 2:

b = b + " and bananas"; //Step 3:

System.out.println("A: " + a);
//A: apples

System.out.println("B: " + b);
//B: apples and bananas
```

- Both a and b point to the same data after the assignment on the second line
- But the address stored in b was automatically changed when the concatenation occurred

String Aliasing

```
String a = "apples"; //Step 1:

String b = a; //Step 2:

b = b + " and bananas"; //Step 3:

System.out.println("A: " + a);
//A: apples

System.out.println("B: " + b);
//B: apples and bananas
```



Section 3

Two-Dimensional Arrays

Storing Temperatures

Let's try to store temperatures taken each week in a year.

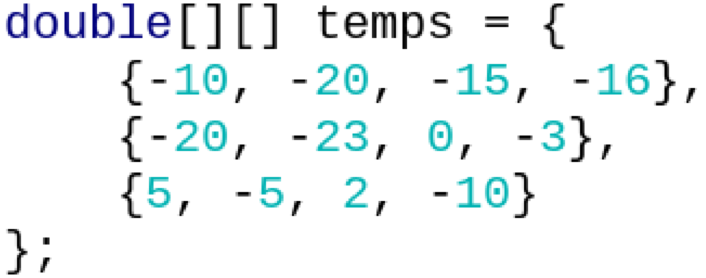
```
double[] janTemps = {-10, -20, -15, -16};
double[] febTemps = {-20, -23, 0, -3};
double[] marchTemps = {5, -5, 2, -10};

//print out Jan temperatures
for (int i=0; i < janTemps.length; i++){
    System.out.print(janTemps.length);
}
System.out.println();
```

- To do this for every month is tedious
- We would have to copy and paste a lot
- Repeated code is bad, because it's easy to make mistakes

2D Arrays

Let's store the weather for the first three months in the same variable
This will be an array of arrays - a *two-dimensional array*

- The inner arrays will store the weather for a particular month
- The outer array stores the inner arrays
- This is an *array of arrays which hold doubles*



```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10}
};
```

Month Weather

```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10}
};

for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];
    System.out.println(Arrays.toString(monthArr));
}
//prints
//[-10.0, -20.0, -15.0, -16.0]
//[-20.0, -23.0, 0.0, -3.0]
//[5.0, -5.0, 2.0, -10.0]
```

- We have a for-loop to go through the months
- Each inner-array is printed out using `Arrays.toString`

Section 4

Creating 2D Arrays

Arrays of Arrays

Three main ways to create arrays of arrays:

1. `int[][] a = {{1, 2, 3}, {5, 6}}`
2. `int[][] b = new int[4][5];`
3. `int[][] c = new int[3][];`

Hard-coding Values

```
int[][] a = {{1, 2, 3}, {5, 6}}
```

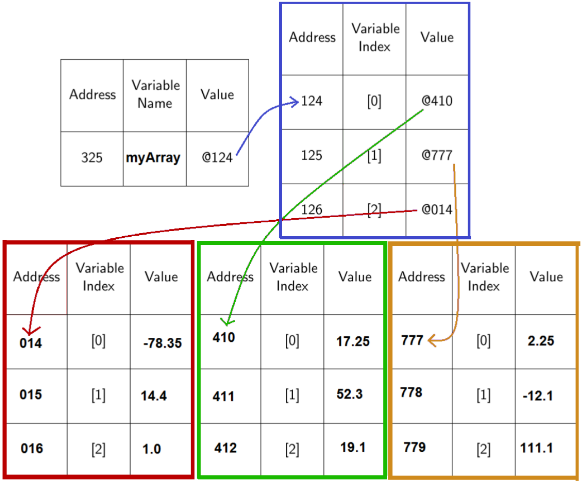
- Creates an array of length 2
- The first element in the array is another array of length 3 (with values 1,2,3).
- The second element is an array of length 2 (with values 5,6).

Arrays as Reference Types

```
double[][] myArray = { {17.25, 52.3, 19.1},
            {2.25, -12.1, 111.1},
            {-78.35, 14.4, 1.0} };
```

Let's see what this looks like in terms of reference types.
That is, how many addresses do we have here?

Address Diagram



Address	Variable Name	Value
325	**myArray**	@124

Address	Variable Index	Value
124	[0]	@410
125	[1]	@777
126	[2]	@014

Address	Variable Index	Value
014	[0]	**-78.35**
015	[1]	**14.4**
016	[2]	**1.0**

Address	Variable Index	Value
410	[0]	**17.25**
411	[1]	**52.3**
412	[2]	**19.1**

Address	Variable Index	Value
777	[0]	**2.25**
778	[1]	**-12.1**
779	[2]	**111.1**

```
double[][] myArray = { {17.25, 52.3, 19.1},
                {2.25, -12.1, 111.1},
                {-78.35, 14.4, 1.0} };
```

Creating An Empty Array

Recall the default value placed in arrays when created:

- int/double: 0
- boolean: false
- char: special value
- Reference types: **null**

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

Creates an array of length 4, where all elements are initialized to 0.

$$0, 0, 0, 0$$

Creating An Empty Array of Arrays

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

- Creates an array of length 2, where each element in that array is another array of length 3
- Because the array store `integers`, every entry will start off with a value of 0

The array will contain `[ [0, 0, 0], [0, 0, 0] ]`

Creating an Empty Array of Arrays

```
int[][] arr = new int[2][];
```

- This creates an array of length 2.
- Each element in the array will be an array.
- However, what is currently stored is the value *null*
  - Represents an uncreated array

The array will contain `[ null, null ]`

Section 5

Printing 2D Arrays

Import Statements

```
import java.util.Arrays;

public class MultiDimInput{
    public static void main(String[] args){
        int[][] a = {{1, 5, 6}, {3,4, 10}};

        System.out.println(Arrays.toString(a));
        //prints [[I@71ba4236, [I@153b2096]

        System.out.println(Arrays.deepToString(a));
        //prints [[1, 5, 6], [3, 4, 10]]
    }
}
```

- `Arrays.deepToString(arr)` $\rightarrow$ prints all of the elements of `arr`, where `arr` is a multi-dimensional array

As before, you may be asked to write the code for this methods on an exam

Month Weather

```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10}
};

for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];
    System.out.println(Arrays.toString(monthArr));
}
//prints
//[-10.0, -20.0, -15.0, -16.0]
//[-20.0, -23.0, 0.0, -3.0]
//[5.0, -5.0, 2.0, -10.0]
```

- We have a for-loop to go through the months
- Each inner-array is printed out using `Arrays.toString`

Deep Printing

```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10}
};

for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];

    for (int dayIndex = 0; dayIndex < monthArr.length; dayIndex++){
        System.out.print(monthArr[dayIndex] + ", ");
    }
    System.out.println();
}
//prints
//-10.0, -20.0, -15.0, -16.0,
//-20.0, -23.0, 0.0, -3.0,
//5.0, -5.0, 2.0, -10.0,
```

Null in Arrays

```
int[][] a = {
    {1, 2, 5}, null
};
System.out.println(Arrays.deepToString(a));
//prints [[1, 2, 5], null]
```

- Note that it's possible to have `null` inside an array of arrays
- This represents an uninitialized element

Null in Arrays

```
double[][] temps = {
    {-10, -20, -15, -16},
    {-20, -23, 0, -3},
    {5, -5, 2, -10},
    null
};

for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];

    if (monthArr == null){
        System.out.println("null value!");
        continue;
    }

    for (int dayIndex = 0; dayIndex < monthArr.length; dayIndex++){
        System.out.print(monthArr[dayIndex] + ", ");
    }
    System.out.println();
}
//prints
//-10.0, -20.0, -15.0, -16.0,
//-20.0, -23.0, 0.0, -3.0,
//5.0, -5.0, 2.0, -10.0,
//null value!
```

- You can test to see if a reference type is `null`
- It's okay to use == here because we are testing to see if the address is `null`

Section 6

Changing Elements in a 2D Array

2D Arrays

```
//create the array
int[][] arr = {{1,2,3},{4,5,6}};

int[] first = arr[0];
int x = first[1];

int y = arr[1][2];
System.out.println(x + " " + y);
//what prints?
```

2 6

The second element of the first array, and the third element of the second array

Inputting Elements in the Array

```
int[][] arr = new int[3][];

System.out.println(Arrays.deepToString(arr));
//prints [null, null, null]

int[] x = {1, 5, 7};
arr[0] = x;

System.out.println(Arrays.deepToString(arr));
//prints [[1, 5, 7], null, null]
```

- Can change elements in a 2D array
- Note that the elements in the outer array are arrays themselves

Inputting Elements in the Array

```java
int[][] arr = new int[3][];

System.out.println(Arrays.deepToString(arr));
//prints [null, null, null]

int[] x = {1, 5, 7};
arr[0] = x;

System.out.println(Arrays.deepToString(arr));
//prints [[1, 5, 7], null, null]

arr[0][1] = 99;
System.out.println(Arrays.deepToString(arr));
//prints [[1, 99, 7], null, null]
```

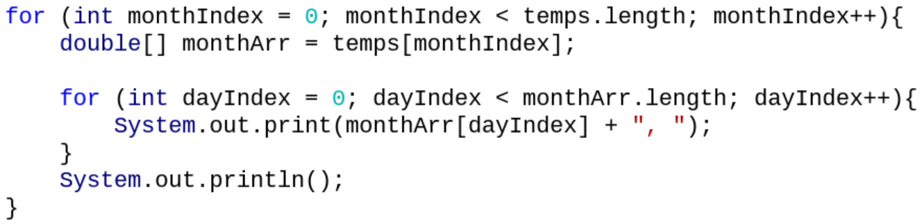
- Can index into the outer and the inner array at once to change elements

Jagged Arrays

`int[][] arr = {{1, 2, 3 }, {0, 0}, {5} }`

The array contains three arrays, of size 3, 2, and 1.

- Sometimes, we may have arrays of arrays of different length
- Be careful on your for-loops that you don't make assumptions about the inner array's length



```
for (int monthIndex = 0; monthIndex < temps.length; monthIndex++){
    double[] monthArr = temps[monthIndex];

    for (int dayIndex = 0; dayIndex < monthArr.length; dayIndex++){
        System.out.print(monthArr[dayIndex] + ", ");
    }
    System.out.println();
}
```

Higher Dimensional Arrays

You can create higher dimensional arrays:

```
int[][][][][][][][][][] x = new
int[2][2][2][2][2][2][2][2][2][2];
```

Anything larger than 3D is very rare
We'll only use 2D in this course

Higher Dimensional Arrays

If we had a method that takes as input an `int[][][]` and returns the largest value in the array:
How many loops do you need to do this?

Answer: Three nested loops
One for the outer array, one for the middle array, and one for the inner array

Multi-Dimensional Arrays

```
int[][][] arr = {
    {
        {1,2,3},{4,5,6},{5,5,5},{1,9,0}
    },
    {
        {3,4,8},{9,1,2},{9,5,1},{3,4,3}
    }
};
System.out.println(arr.length);          //2
System.out.println(arr[0].length);       //4
System.out.println(arr[0][0].length);    //3
System.out.println(arr[1][2][0]);        //9
```

Section 7

2D Array Examples

Sum All Numbers

Write a method that takes as input a 2D array of doubles and returns the sum of all of the numbers in the array.

Sum All Numbers

```
public static void main(String[] args){
    double[][] arr = {
        {134.6, 1235.2, 1314.5},
        {1934, 134.1, 13923.4324, 434},
        {1323, 1},
        {},
        {34343,234,24,2426,47,47}
    };

    double sum = getSum(arr);
    System.out.println("Sum: " + sum);
}

public static double getSum(double[][] arr){
    double sum = 0;
    for (int outer = 0; outer < arr.length; outer++){
        for (int inner = 0; inner < arr[outer].length; inner++){
            sum += arr[outer][inner];
        }
    }
    return sum;
}
```

Swapping Arrays

```
public static void main(String[] args){
    int[][] arr = {{1,2},{3,4}};
    //swap the two inner arrays
    swap(arr, 0, 1);
    //print out the inner arrays
    System.out.println(Arrays.toString(arr[0]) + " " +
                       Arrays.toString(arr[1]));
}
//pass the array of arrays, and the two indices
public static void swap(int[][] b, int i, int j){
    //switch the arrays
    int[] temp = b[i];
    b[i] = b[j];
    b[j] = temp;
}
```

What prints?

```
[3, 4] [1, 2]
```

Selecting a Column

Write a method that takes as input a 2D integer array and an integer number representing a column index. It should return the column at that index.
For example, if the input array is:

```
int[] arr = {
{1,2,3},
{5,6,7},
{9,2,1},
{0,3,0}
};
```

And the index is 1, it should return the array. `{2,6,2,3}`

columnSelect

```
public static int[] columnSelect(int[][] arr, int col){

    //figure out how many entries are in the column
    int numEntries = arr.length;
    int[] result = new int[numEntries];

    //go through the arr, for each sub-array,
    //and select each entry
    for (int i=0; i < numEntries; i++){
        result[i] = arr[i][col];
    }
    return result;
}
```

Multiply by a Value

Write a method `multiplyMatrix` that takes as input one 2D array representing a matrix, as well as a double value. Multiply each element in the array by that value and do not return anything.

multiplyMatrix

```
public static void main(String[] args){
    double[][] matrix = {
        {1,4,5},
        {4,9,5},
        {3.4,5.4,34}
    };

    System.out.println(Arrays.deepToString(matrix));
    multiplyMatrix(matrix, 3.4);
    System.out.println(Arrays.deepToString(matrix));
}

public static void multiplyMatrix(double[][] mx, double val){

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

        double[] innerMx = mx[i];
        for (int j = 0; j < innerMx.length; j++){
            innerMx[j] *= val;
        }
    }
}
```

Practice

Write `multiplyMatrix` again, but make a copy of the input, multiply each element of the copy, and return that instead. Do not modify the original.

Be sure to copy at the level of primitive types! (Don't copy any references)
Throw an exception if the input is not rectangular, or if any of the 'sub-arrays' are null.

sumMatrix

Write a method sumMatrix that takes two 2D integer arrays as input with the same dimensions. The method should return a new 2D integer array corresponding to their sum.

Example: consider the following 2D arrays

```
int[][] matrix1 = {{2,3}, {5,1}};
int[][] matrix2 = {{-1,5}, {2,-4}};
```

Then sumMatrix(matrix1, matrix2) should return the 2D array

```
{{1, 8}, {7, -3}}
```

sumMatrices

```java
public static int[][] sumMatrices(int[][] mx1, int[][] mx2){

    int size = mx1.length;

    int[][] result = new int[size][size];

    for (int i = 0; i < size; i++){
        for (int j = 0; j < size; j++){
            result[i][j] = mx1[i][j] + mx2[i][j];
        }
    }
    return result;
}
```

Practice

Write `sumMatrices` again, but make a copy of the first array and add each element of the second array to the copy, and return that instead. Do not modify the original.

Be sure to copy at the level of primitive types! (Don't copy any references) Throw an exception if any of the 'sub-arrays' are null.