Arrays

CSE 1310 – Introduction to Computers and Programming
University of Texas at Arlington
DEPARTMENTAL FINAL EXAM
Monday, Dec 9, 5:30pm-8pm

rooms will be determined
Motivation

• Current limitation: We cannot ‘record’ multiple items as a group (e.g. grades).
  – Currently we can only do that by having a different variable for each item (e.g. grade). That is “not correct programming”.
    • A program must be a solution to all problems of a certain type, not to a specific instance. E.g. any program working with a set of values (e.g. student grades) should not hardcode the number of grades. It must be able to work with any number of grades without having it’s code modified.

• Solution: collections (of 2 types)
  – They can store a set of values of the same type
  – We can access the values with an index (like for strings)
  – 2 types of collections:
    • Arrays – must specify their size; hard to insert and delete from them
    • ArrayLists – they can readjust their size as needed, and it is easy to insert and delete from them. Covered in future lecture.
Examples of problems we need collections for

• Given a set of numbers, count those smaller than the last one:
  – Ask the user to enter N numbers.
    • Or the user enters ‘q’ to stop.
  – Print how many of those numbers are less than the last number entered.
    • Note the difference between this problem and counting number of values smaller than the first value which does not require ‘storing’ numbers.

• Phone book:
  – Ask the user to specify an integer N.
  – Ask the user to enter N names and phone numbers.
  – Then, whenever the user types a name, the computer outputs the corresponding phone number.

• Grade roster:
  – Read in k homework grades for N students (N and k provided by the user)
  – Compute and print average for each homework and average for each student

• Printing calendar months and their names
Collections: Operations and Processing

• Basic operations
  – Declare and
    • Create and initialize with an initial set of values.
    • Create empty (with no values. We may have to declare the size.)
  – Iterate over all the elements/items/values
    • E.g. to print them or inspect them
  – Modify an element
  – Add an element
    • At the end
    • Between other elements
  – Remove an element

• Processing
  – Find smallest/largest
  – Compute sum/average
  – Search for a value
  – Print elements with separators between them (like a table)

• 1D vs 2D (‘table’/’matrix’) collections
  – 1D = a row or a vector
  – 2D = a table or a matrix
    • 2D collections can be mixed (e.g. array of ArrayList)
More on collections

- 2-dimensional (‘table of elements’)
- Making a copy of a collection
- Collections used in method communication:
  - as arguments to methods.
    - OK: modify the content of the collection
    - NO: modify the collection reference
  - Returned by the method
Terminology

- Collection
- Array
- ArrayList
- Element/item in a collection
- Reference:
  - Array reference, object reference
- Iterate over a collection
- Modify/insert/remove from a collection
- Copy vs 2 references to the same collection
- Empty collection
- Modify content vs change the reference
- 1D array, 2D array
A First Example

• Printing months and their lengths.
• With arrays:
  – One array variable for month names.


  – One array variable for month lengths.

A First Example

• Printing out months and lengths is easy, using a loop.

```java
public class months_arrays {
    public static void main(String[] args) {
        String[] month_names = {"January", "February", "March",
            "April", "May", "June",
            "July", "August", "September",
            "October", "November", "December"};

        int[] month_lengths = {31, 28, 31, 30, 31, 30,
            31, 31, 30, 31, 30, 31};

        for (int i = 0; i < 12; i++)
        {
            System.out.printf("%s has %d days.\n",
                month_names[i], month_lengths[i]);
        }
    }
}
```
Arrays Simplify Code

• Entering data remains painful.
  – Either way we must enter 12 names and 12 lengths.
  – The solution to that will be **files** (our next topic).
    • Data will be read automatically from files.

• Manipulating data becomes much easier.
  – We can go through data using loops.
  – We can process **millions of items** (strings, numbers) with few lines of code.
  – Changes (e.g. print in a different way) only require changing a few lines of code in the loop and they apply to all the items.
Creating an Array of Numbers

• One way: providing an initial list of values.

```java
int[] numbers_1 = {10, 2, 5, 40, 30, 100, 200};
```

• Another way: providing just the length.
  – This version initializes all values to 0.

```java
int[] numbers_2 = new int[4];
```
Creating an Array of Strings

• Providing an initial list of values.

```java
String[] names = {"Mary", "Ann", "Joe"};
```

- Memory address: 3000
- "Mary" | "Ann" | "Joe"

• Providing just the length.
  – This approach initializes all values to **null**.
    • **null** means “not a valid memory address”, which here means NOT valid strings.

```java
String[] names = new String[6];
```

- Memory address: 5000
- null | null | null | null | null | null
Accessing Single Elements

```java
int[] numbers = {10, 2, 5, 40, 30, 100, 200};
int a = numbers[0];
System.out.printf("%d", numbers[5]);
```

• The above code:
  – creates an array of 7 integers.
  – numbers[0] refers to element at index 0 in the array, which is 10.

**IMPORTANT: ELEMENT POSITIONS START WITH 0, NOT WITH 1.**

  – numbers[5] refers to element at index 5 in the array, which is 100.
Accessing Single Elements

```java
int[] numbers = {10, 2, 5, 40, 30, 100, 200};
int a = numbers[7];
```

- The above code will do what?
Accessing Single Elements

```java
int[] numbers = {10, 2, 5, 40, 30, 100, 200};
int a = numbers[7];
```

- The above code will crash.
  - `numbers[7]` **does not exist**, valid positions are only from 0 to 6.
  - Produces an IndexOutOfBoundsException
Length of an Array

```java
int[] numbers = {10, 2, 5, 40, 30, 100, 200};
for (int i = 0; i < numbers.length; i++)
{
    System.out.printf("%d\n", numbers[i]);
}
```

- The above code prints all 7 elements of the array.
- `numbers.length` gives the number of elements in the array.
Changing Single Elements

```java
int[] numbers = {10, 2, 5, 40, 30, 100, 200};
numbers[0] = 3;
numbers[4] = 15;
for (int i = 0; i < numbers.length; i++){
    System.out.printf("%d\n", numbers[i]);
}
```

Output:
```
3
2
5
40
15
100
200
```

```java
String[] str = new String[5];
str[2] = "Chicago";
for (int i = 0; i < str.length; i++) {
    System.out.printf("%s\n", str[i]);
}
```

Output:
```
null
null
Chicago
New York
null
```
Read an Array from the User - Program

• Write a program that:
  – Asks the user to enter an integer N.
  – Asks the user to enter N values, and stores them in an array.
  – Prints out the values.

Example Output:

Enter N: 5
Enter value 0: 40
Enter value 1: 10
Enter value 2: 80
Enter value 3: 100
Enter value 4: 20

numbers[0] = 40
numbers[1] = 10
numbers[2] = 80
numbers[3] = 100
numbers[4] = 20
import java.util.Scanner;

public class read_n_numbers {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.printf("Enter N: ");
        int N = in.nextInt();

        int[] numbers = new int[N];
        for (int i = 0; i < N; i++) {
            System.out.printf("Enter value %d: ", i);
            numbers[i] = in.nextInt();
        }

        System.out.printf("\n");
        for (int i = 0; i < N; i++) {
            System.out.printf("numbers[%d] = %d\n", i, numbers[i]);
        }
    }
}

A program that:
• Reads N integers from the user.
• Stores those integers in an array.
• Prints the contents of the array.
Write a function that:

- Asks the user to enter an integer N.
- Asks the user to enter N values, and stores them in an array.
- Returns the array.
Read an Array from the User - Function

- Write a function that:
  - Asks the user to enter an integer N.
  - Asks the user to enter N values, and stores them in an array.
  - Returns the array.

```java
public static int[] user_integers()
{
    Scanner in = new Scanner(System.in);
    System.out.printf("Enter N: ");
    int N = in.nextInt();

    int[] result = new int[N];
    for (int i = 0; i < N; i++)
    {
        System.out.printf("Enter value %d: ", i);
        result[i] = in.nextInt();
    }
    return result;
}
```
Read an Array from the User - Function

- Using our `user_integers` function, the main function looks more simple:

```java
public static void main(String[] args) {
    int[] numbers = user_integers();

    System.out.printf("\n");
    for (int i = 0; i < numbers.length; i++) {
        System.out.printf("numbers[%d] = %d\n", i, numbers[i]);
    }
}
```
Finding the **Smallest** Value

- Write a function `find_min` that:
  - Takes as input an array of integers.
  - Returns the smallest value among those integers.

```java
public static int find_min(int[] values) {
    int result = values[0];
    for (int i = 0; i < values.length; i++) {
        if (values[i] < result) {
            result = values[i];
        }
    }
    return result;
}
```
Finding the **Largest** Value

- Write a function `find_max` that:
  - Takes as input an array of integers.
  - Returns the largest value among those integers.

```java
public static int find_max(int[] values) {
    int result = values[0];
    for (int i = 0; i < values.length; i++) {
        if (values[i] > result) {
            result = values[i];
        }
    }
    return result;
}
```
An Example Program

• Write a program that:
  – Asks the user to enter an integer N.
  – Asks the user to enter N values, and stores them in an array.
  – Prints out the values, indicating the maximum and the minimum.

Example Output:

Enter N: 5
Enter value 0: 40
Enter value 1: 10
Enter value 2: 80
Enter value 3: 90
Enter value 4: 20

numbers[0] = 40
numbers[1] = 10 *** smallest value ***
numbers[2] = 80
numbers[3] = 90 *** largest value ***
numbers[4] = 20
public static void main(String[] args) {
    int[] numbers = user_integers();
    int smallest = find_min(numbers);
    int largest = find_max(numbers);
    System.out.printf("\n");
    for (int i = 0; i < numbers.length; i++)
    {
        System.out.printf("numbers[%d] = %d", i, numbers[i]);
        if (numbers[i] == smallest)
        {
            System.out.printf(" *** smallest value ***\n");
        }
        else if (numbers[i] == largest)
        {
            System.out.printf(" *** largest value ***\n");
        }
        else
        {
            System.out.printf("\n");
        }
    }
}
import java.util.Arrays;
...

Arrays.sort(arr);  // sorts in increasing order
// no sorting in reverse order.

// easy printing:
String toPrint = Arrays.toString(arr);
System.out.println("Array: " + toPrint);

// make a copy of an array (:
int[] arr_1 = {4,1,9,0,3};
int[] arr_2 = Arrays.copyOf(arr_1, 2*arr_1.length);

Read the documentation: https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

• ==, Arrays.equals(arr1, arr2), Arrays.deepEquals(arr1, arr2) (use 1D and 2D arrays)
• Arrays.toString(arr), Arrays.deepToString(arr) (use 1D and 2D arrays)
• Arrays.sort(arr), Arrays.binarySearch(arr, val),
• Arrays.copyOf(arr, sz), Arrays.copyOfRange(arr, fromIndex, toIndex)
• Arrays.fill(arr, value), Arrays.fill(arr, fromIndex, toIndex, value)
Variables Pointing to Same Set

• This topic is a VERY COMMON SOURCE OF MISUNDERSTANDINGS.

• When two array (or array list) variables are set equal to each other, they are fundamentally linked:
  – They both refer to the same set of values.
    • In computer science, we say that they are both references, pointing to the same set of values.
  – Any modification to that set of values affects all the variables that point to it.
    – It is the same as sharing a document with multiple user and giving them access to modify the document.

• It is important to identify (and treat separately) assignments of array variables vs. modifications.
int[] a = {10, 20, 30, 40};  //line 1
int[] b = a; //shallow copy , line 2
b[2] = 7;
for (int i = 0; i < a.length; i++)
{
    System.out.printf("a[%d] = %d\n", i, a[i]);
}

Output:

Draw picture to show the data.
Sharing of Modifications: Example

int[] a = {10, 20, 30, 40}; //line 1
int[] b = a; //shallow copy, line 2
b[2] = 7;
for (int i = 0; i < a.length; i++)
{
    System.out.printf("a[%d] = %d
", i, a[i]);
}

Output:
- a[0] = 10
- a[1] = 20
- a[2] = 7
- a[3] = 40

The variable for the array (i.e. a holds 65ae6ba4) hold the memory address where the actual data from that array will be stored.
Deep copy: an actual copy of the array data

- Allocate space.
- Copy the elements one by one.

```java
int[] a = {10, 20, 30, 40};  //line 1
//The next two lines make deep copy of a in b
int[] b = new int[a.length];  // allocate space, line2
// end of deep copy code
b[2] = 7;
for (int i = 0; i < a.length; i++) {
    System.out.printf("a[%d] = %d\n", i, a[i]);
}
```

Output:
- `a[0] = 10`
- `a[1] = 20`
- `a[2] = 30`
- `a[3] = 40`
- `b[2] = 7`
Another Example

What will this program print?

Output:

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;
        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
        {
            System.out.printf("a[%d] = %d\n", i, a[i]);
        }
        for (int i = 0; i < c.length; i++)
        {
            System.out.printf("c[%d] = %d\n", i, c[i]);
        }
    }
}
```
Another Example

What will this program print?

Output:

a[0] = 10
a[1] = 20
a[2] = 7
a[3] = 40
c[0] = 4
c[1] = 15
c[2] = 2

public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);

        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}

See line-by-line execution
- with debugger
- at the end of the slides
Variables Pointing to Same Set (Recap)

• When two array (or array list) variables are set equal to each other, they point to the same underlying array (or array list):
  – Any modification to values of that array (or array list) affects all the variables that point to it.
• The only way to break the link between two array (or array list) variables, is to assign an array (or array list) variable to some other value.
• Given a line of code involving an array (or array list) variable, we should be able to identify:
  – Does this line assign a value to the array (or array list) variable?
  – Does this line simply modify data at one or more positions in the array?
• These two cases are different, and follow different rules.
foo takes arguments of types array and int and makes changes to them. Pay attention to which of those changes remain after the call to foo ends and which exist only during foo.

```
public class example1 {
    public static void foo(int[] x, int[] y, int N) {
        x[2] = 0; // will persist
        y = new int[17]; // will not persist
        N = 0; // will not persist
    }
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = {1, 2, 3, 4, 5, 6};
        int N = 121;
        foo(a, b, N);
        System.out.printf("a[2] = %d\n", a[2]);
        System.out.printf("b.length = %d\n", b.length);
        System.out.printf("N = %d\n", N);
    }
}
```

Changes to data in array a persisted after call to foo ended.
Changes to address of array b did NOT persist after call to foo ended.
Changes to value of int N did NOT persist after call to foo ended.
This behavior is consistent since variables a and b hold the address of the arrays, not the actual data.
DRAW data after each step.
The split Method

- The **split** method creates an array of smaller strings from a larger string based on one or more *separators*.
- split is called on the longer string (the one to “to be cut”)
- Takes a single argument: a string with the separator(s).
- Returns the created array

```java
String text = "Today is a hot summer day.";
String[] words = text.split(" ");
for (int i = 0; i < words.length; i++) {
    System.out.printf("word[%d] = %s\n", i, words[i]);
}
```

Output:
- words[0] = Today
- words[1] = is
- words[2] = a
- words[3] = hot
- words[4] = summer

"Today is a hot summer day.".split(" ")

Here it separates at an empty space
The split Method

• You can specify multiple characters that separate words.
• Below, 4 separators are given: comma, space, and dash.
• To specify multiple characters, you must enclose them in square brackets: [] (see the example argument: "[, -]").

```java
String text = "Let's count: One,two,three.";
String[] words = text.split("[, -]");
for (int i = 0; i < words.length; i++) {
    System.out.printf("word[%d] = %s\n", i, words[i]);
}
```

Output:
word[0] = Let's
word[1] = count:
word[2] = One
word[3] = two

• If no [], then the entire argument string is the separator. See "two":

```java
String text = "Let's count: One,two,three.";
String[] words = text.split("two");
for (int i = 0; i < words.length; i++) {
    System.out.printf("word[%d] = %s\n", i, words[i]);
}
```

Output:
word[0] = Let's count: One,
word[1] = ,three.
Split method

This method is useful for

• reading easily and array of strings from the user
• Extracting individual pieces of information from a string in a specific format. E.g. :
  – date (09/15/2019):
    "09/15/2019".split("/")  -> ["09 " ,"15", "2019"]
  – phone number: 817-888-9999
    "817-888-9999".split("-") -> ["817 " ,"888", "9999"]

See also how to convert a string to a number (int or double):

```java
int x = Integer.parseInt("78"); // throws error if bad
int bad = Integer.parseInt("78.3"); // throws NumberFormatException
double y = Double.parseDouble("78.3");
Integer n = Integer.valueOf("78");  // valueOf returns Integer object
```
You can have arrays of arrays.
- These are called 2-dimensional arrays, or matrices or tables.
- You can have arrays of arrays of arrays of arrays ...

double[][] a = { {3.2, 2.1, 5.3},
                {8.0, 4.9, 5.7} };
a[1][0] = 2;
System.out.printf("%.1f\n", a[0][0]);
System.out.printf("%.1f\n", a[1][0]);
System.out.printf("%.1f\n", a[1][1]);

Simple representation

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.2</td>
<td>2.1</td>
<td>5.3</td>
</tr>
<tr>
<td>8.0</td>
<td>4.9</td>
<td>5.7</td>
</tr>
</tbody>
</table>

Actual representation in memory:
a holds the memory address of an array with 2 other memory address (one for each row)
# 2D arrays

Remember: A 2D array is a 1D array of 1D arrays

<table>
<thead>
<tr>
<th>Code</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int[][] table1 = new int[3][5]; // all elements are 0</code></td>
<td>2D array (table) with 3 rows, 5 columns</td>
</tr>
<tr>
<td><code>int[][] table = { {10,15,0,2,8}, {4,8,9,6,7}, {1,2,3,7,5} };</code></td>
<td>Create and initialize table</td>
</tr>
<tr>
<td><code>int[][] zigzag = { {1,2,3,4,5}, {10,15}, {9,4,1,8}, {6} };</code></td>
<td>Table where rows have different length</td>
</tr>
<tr>
<td><code>table.length</code></td>
<td>Number of rows</td>
</tr>
<tr>
<td><code>table[0].length</code></td>
<td>Number of elements on row 0</td>
</tr>
<tr>
<td><code>table[r].length</code></td>
<td>Number of elements on row r</td>
</tr>
<tr>
<td><code>table[1][3]</code></td>
<td>Element at row 1, column 3</td>
</tr>
<tr>
<td><code>table[r][c]</code></td>
<td>Element at row r column c</td>
</tr>
<tr>
<td>Need nested loop to iterate over whole table</td>
<td></td>
</tr>
</tbody>
</table>
2D array traversals and element access

Indexes of neighboring elements of element at row $r$ and column $c$:

<table>
<thead>
<tr>
<th></th>
<th>c-1</th>
<th>c</th>
<th>c+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>r-1</td>
<td>[r-1][i-1]</td>
<td>[r-1][i]</td>
<td>[r-1][i+1]</td>
</tr>
<tr>
<td>r</td>
<td>[r][i-1]</td>
<td>[r][i]</td>
<td>[r][i+1]</td>
</tr>
<tr>
<td>r+1</td>
<td>[r+1][i-1]</td>
<td>[r+1][i]</td>
<td>[r+1][i+1]</td>
</tr>
</tbody>
</table>

Practice: For each of the showed type of data (neighbors, diagonals, row, column) write a method that takes a matrix and other needed info and prints that data.

**First diagonal**

<table>
<thead>
<tr>
<th></th>
<th>Second diagonal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,0</td>
<td>0,4</td>
</tr>
<tr>
<td>1,1</td>
<td>1,4</td>
</tr>
<tr>
<td>2,2</td>
<td></td>
</tr>
<tr>
<td>3,1</td>
<td>3,3</td>
</tr>
<tr>
<td>4,0</td>
<td>4,4</td>
</tr>
</tbody>
</table>

**Column**

<table>
<thead>
<tr>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,c</td>
</tr>
<tr>
<td>1,c</td>
</tr>
<tr>
<td>2,c</td>
</tr>
<tr>
<td>r,0</td>
</tr>
<tr>
<td>r,c</td>
</tr>
<tr>
<td>r,2</td>
</tr>
<tr>
<td>r,3</td>
</tr>
<tr>
<td>r,4</td>
</tr>
<tr>
<td>4,c</td>
</tr>
</tbody>
</table>
2D Arrays operations

- In a 2D array rows are arrays (and all the issues with normal arrays apply: uninitialized...)
- Find size: rows and columns
- Declare, access elements, modify elements
- Locating neighboring elements (6.7.3): Indexes and indexes of neighboring elements (the 3x3 table of i,j indexes)
- Operations over rows and columns: sum
- Using a 2D array as a parameter for a method
- Self check nice exercises:
  - Generate checker board (alternating 0/1)
  - Tic-tac-toe: identify indexes for first and second diagonal, (also a row and a column reviewed)
- Special topic:
  - 2D arrays with variable row length (e.g. triangle shaped array): allocate the array of rows first and then allocate each row.
  - Multidimensional array (e.g. 3D)
Practice problem: Magic square

- A **magic square** is an $n \times n$ matrix filled with numbers 1,2,...,n\*n such that the following sums are equal. Sum of the elements in each
  - row,
  - column
  - 1\textsuperscript{st} diagonal and
  - 2\textsuperscript{nd} diagonal.

- Write a program that reads (from the user) numbers for a 4x4 matrix and checks if it is a magic square:
  - Values: 1,2,...,16 (no other numbers allowed)
  - Sums over rows, columns, diagonals are equal.

- Example:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>3</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>11</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>14</td>
<td>1</td>
</tr>
</tbody>
</table>

Generate Magic Squares: [http://www.dcode.fr/magic-square](http://www.dcode.fr/magic-square)
Do you know how to...

• Create an array with hardcoded data
• Create an ‘empty’ array of a specific length and populate it with data from the user
• Print an array (using a loop, not Java magic with Arrays.copyOf() )
• Make a copy of an array
• Modify an element at a specific position
• Swap two elements (given their position) in an array
• Swap two elements given their value (e.g. swap "Jane" and "Mary“)
• Create a new array based on data from given array (e.g. squares, lengths, averages)
• Reverse an array (by swapping pairs of elements).
• Insert a new item in an array
• Remove an item from an array

• All these for arrays of any type (int, double, String, boolean)
• (All these for ArrayList as well. To be covered shortly.)
Practice Problems

• Shift an array by k positions to the (left or to the right). (Start with special method for k = 1.)

• Given array of strings and array of int, find the smallest string from the strings at positions given by the array of int.

• Given an array of strings return an array of int with the lengths of the corresponding strings.
  – Dress up this problem or create another dressed up problem with a similar type of work.
Line-by line execution
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);
        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}

Output:
Another Example

What will this program print?

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);
        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}
```

Output:

```
a[0] = 10
a[1] = 20
a[2] = 7
a[3] = 40
c[0] = 4
c[1] = 15
c[2] = 2
```

See line-by-line execution
- with debugger
- In following slides
Another Example

• Line-by-line execution

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
        {
            System.out.printf("a[%d] = %d\n", i, a[i]);
        }
        for (int i = 0; i < c.length; i++)
        {
            System.out.printf("c[%d] = %d\n", i, c[i]);
        }
    }
}
```

Variables:

```
a ➔
```
Another Example

- Line-by-line execution
  - Assignment of array variable.

Variables:

\[ \text{a} \rightarrow \{10, 20, 30, 40\} \]

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++) {
            System.out.printf("a[%d] = %d\n", i, a[i]);
        }
        for (int i = 0; i < c.length; i++) {
            System.out.printf("c[%d] = %d\n", i, c[i]);
        }
    }
}
```
Another Example

- Line-by-line execution
  - Assignment of array variable.

Variables:

```
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);

        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}
```

Understanding this line is the key:

We should NOT represent this as:

```
a = {10, 20, 30, 40}
b = {10, 20, 30, 40}
```

Example
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;
        b = c;
        b[1] = 15;
        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);
        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}

Variables:

• Line-by-line execution
  – Assignment of array variable.

a → {10, 20, 30, 40}
b
 c → {4, 3, 2}
Another Example

- Line-by-line execution
  - Modification of array data.

Variables:

- $a \rightarrow \{10, 20, 7, 40\}$
- $b$
- $c \rightarrow \{4, 3, 2\}$

Since $a$ and $b$ point to the same DATA, it is clear that changing $b$ changes $a$ at the same time.

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
        {
            System.out.printf("a[%d] = %d\n", i, a[i]);
        }

        for (int i = 0; i < c.length; i++)
        {
            System.out.printf("c[%d] = %d\n", i, c[i]);
        }
    }
}
```
Another Example

• Line-by-line execution
  – Assignment of array variable.

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);

        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}
```

Variables:

- `a` → `{10, 20, 7, 40}`
- `b` → `{4, 3, 2}`
- `c` → `{4, 3, 2}`

Again, we should NOT represent this as:

- `b = {4, 3, 2}`
- `c = {4, 3, 2}`
Another Example

- Line-by-line execution
  - Modification of array data.

```java
public class assignments {
    public static void main(String[] args) {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
            System.out.printf("a[%d] = %d\n", i, a[i]);

        for (int i = 0; i < c.length; i++)
            System.out.printf("c[%d] = %d\n", i, c[i]);
    }
}
```

Variables:

- `a` → `{10, 20, 7, 40}`
- `b` → `{4, 15, 2}`

Since `b` and `c` point to the same array, it is clear that changing `b` changes `c` at the same time.
Important

- When you assign a value to an array (or array list) variable, you create new arrows, or change where the arrows point.
- When you modify an array (or array list), the arrows are not changed.

```java
public class assignments {
    public static void main(String[] args)
    {
        int[] a = {10, 20, 30, 40};
        int[] b = a;
        int[] c = {4, 3, 2};
        b[2] = 7;

        b = c;
        b[1] = 15;

        for (int i = 0; i < a.length; i++)
        {
            System.out.printf("a[%d] = %d\n", i, a[i]);
        }

        for (int i = 0; i < c.length; i++)
        {
            System.out.printf("c[%d] = %d\n", i, c[i]);
        }
    }
}
```

Variables:

- `a` → `{10, 20, 7, 40}`
- `b` → `{4, 15, 2}`