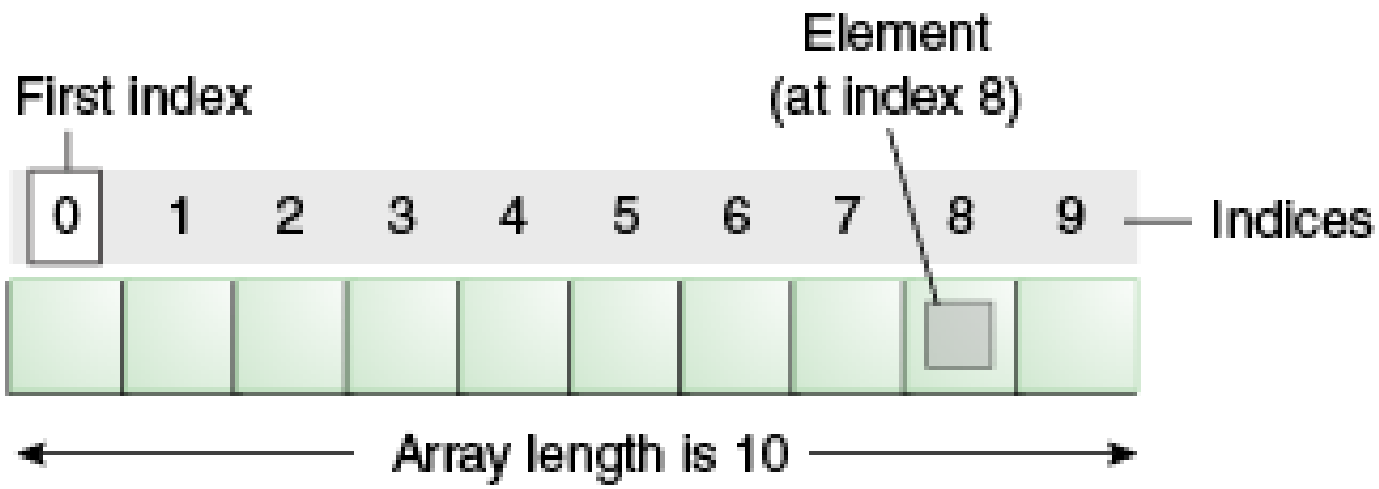
Java Arrays



Computer Science 112
Boston University

Christine Papadakis

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal1(int[] values) {
        int min = values[0];
        for (int i = 0; i < values.length; i++) {
            if (values[i] < min) {
                min = values[i];
            }
        }
        return min;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int min = minVal1(values);
        System.out.println("the min value is " + min);
    }
}
```

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal1(int[] values) {
        int min = values[0];
        for (int i = 1; i < values.length; i++) {
            if (values[i] < min) {
                min = values[i];
            }
        }
        return min;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int min = minVal1(values);
        System.out.println("the min value is " + min);
    }
}
```

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal1(int[] values) {
        int min = values[0];
        for (int i = 1; i < values.length; i++) {
            if (values[i] < min) {
                min = values[i];
            }
        }
        return min;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int min = minVal1(values);
        System.out.println("the min value is " + min);
    }
}
```

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal1(int[] values) {
        int min = values[0];
        for (int i = 0; i < values.length; i++) {
            if (values[i] < min) {
                min = values[i];
            }
        }
        return min;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int min = minVal1(values);
        System.out.println("the min value is " + min);
    }
}
```

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal1(int[] values) {
        int min = values[0];
        for (int i = 0; i < values.length; i++) {
            if (values[i] < min) {
                min = values[i];
            }
        }
        return min;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int min = minVal1(values);
        System.out.println("the min value is " + min);
    }
}
```

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal
        int min = v
        for (int i
            if
            }
        }
        return min;
    }

    public static void main(String[] ags) {
        int[] values = {7, 8, 9, 6, 10, , 9, 5};

        System.out.println("the min value is " +
                                    minVal1( values) );
    }
}
```

Calling the method `minVal1` from within the `println` statement!

Here's the method in the context of a program…

```
public class ArrayMethods {
    /*
     * minVal1 - uses an index-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal
        int min = v
        for (int i
            if
            }
        }
        return min;
    }

    public static void main(String[] ags) {
        int[] values = {7, 8, 9, 6, 10, , 9, 5};
        minval1(values);
        System.out.println("the min value is " +
                                    minVal1( values) );
    }
}
```

This is wrong, as the method is being invoked twice!

Finding the Smallest Value, a variation

```
public class ArrayMethods {
    /*
     * minVal2 - uses an element-based loop to find
     * and return the smallest value in the array values.
     */
    public static int minVal2(int[] values) {
        int min = values[0];
        for (int val : values) {
            if (val < min) {
                min = val;
            }
        }
        return min;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int min = minVal2(values);
        System.out.println("the min value is " + min);
    }
}
```

What if we wanted to find and return the index of the minimum value of the array?

```
public class ArrayMethods {
    /*
     * minIndex - uses an index-based loop to find and
     * and return the index of the smallest value in values.
     */
    public static int minIndex(int[] values) {
        int minIndex = _______________;
        for (_____________________________________) {
            if (_______________________) {

            }
        }
        return minIndex;
    }

    public static void main(String[] args) {
        int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
        int minInd = minIndex(values);
        System.out.println("min value is at index " + minInd);
    }
}
```

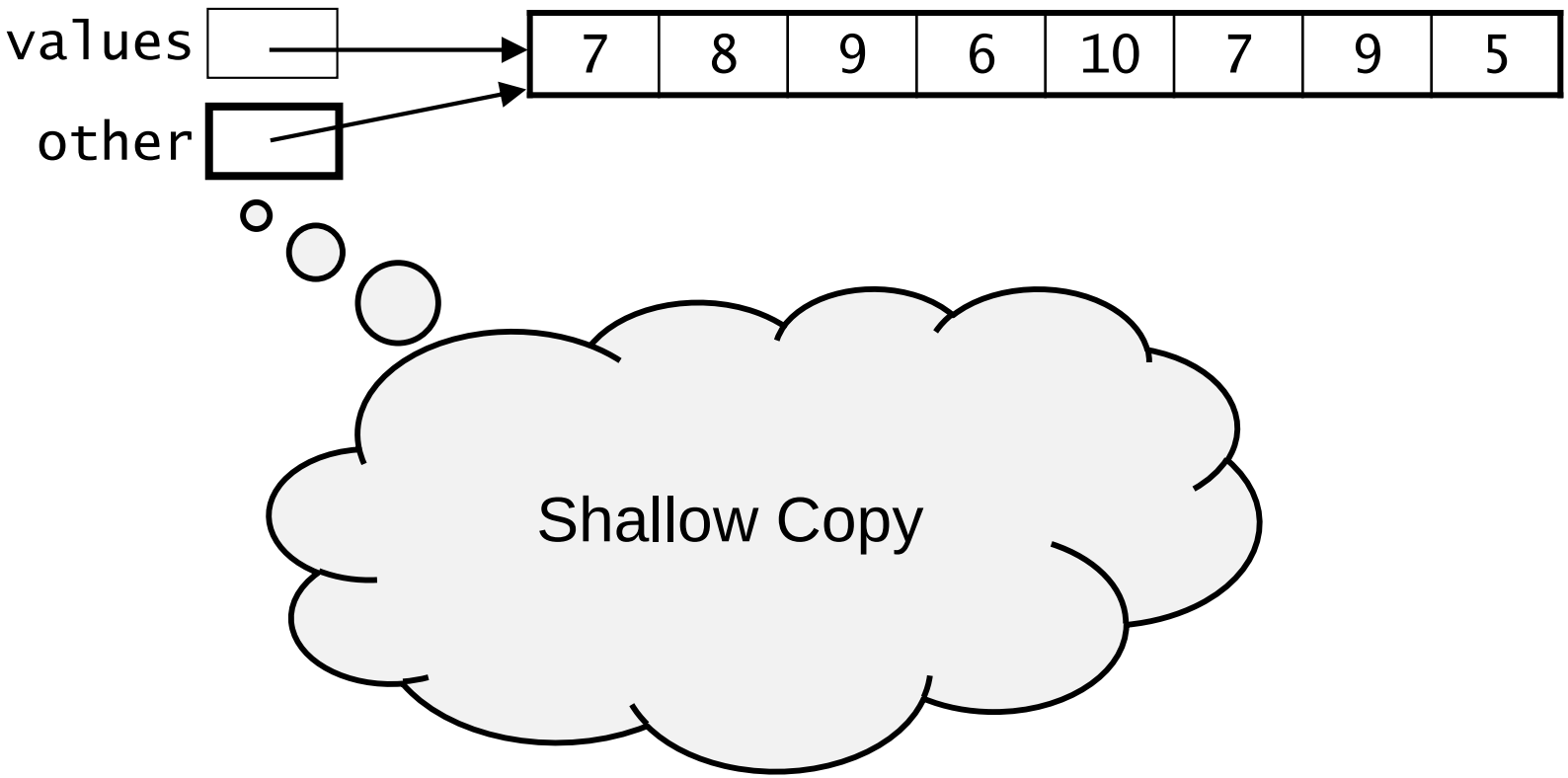
try this later on your own!

Array Assignment

Copying a Reference Variable

- What does this do?

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = values;
```

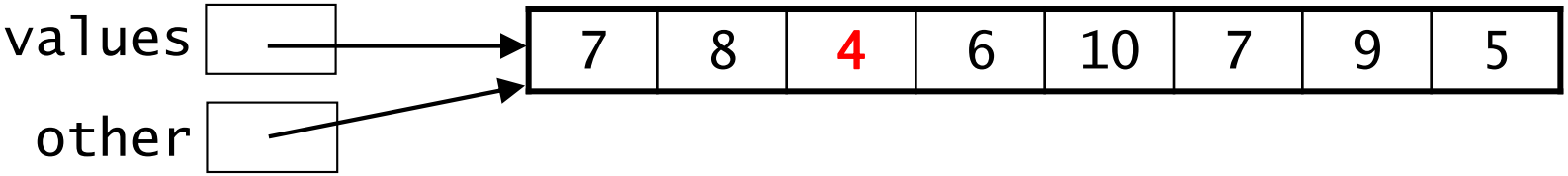


Array Assignment

Copying a Reference Variable

- What does this do?

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = values;
```



- Given the lines of code above, what will the lines below print?

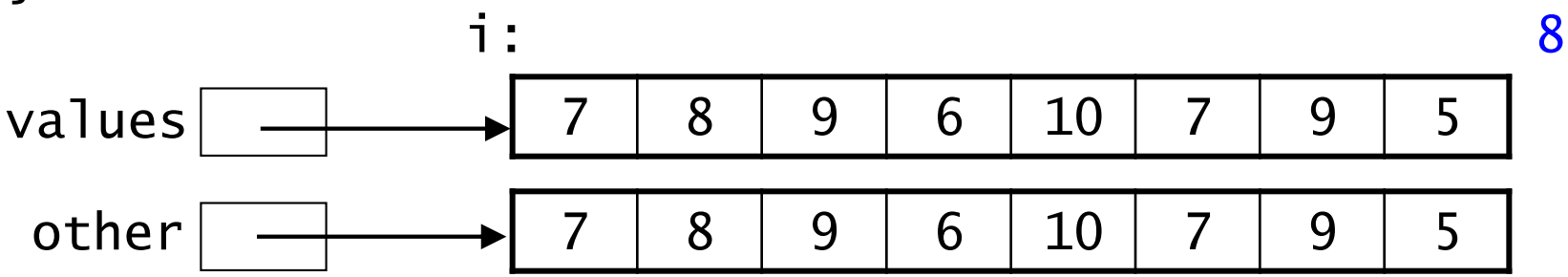
```
other[2] = 4;
System.out.println(values[2] + " " + other[2]);
                   4         + " " +    4
                              "4 4"
```

Copying an Array

- To actually create a copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```



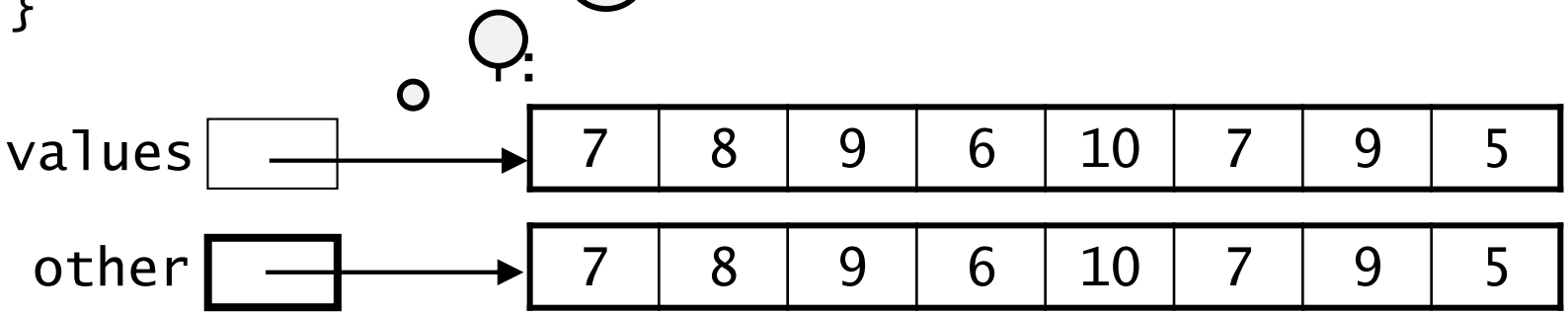
Copying an Array

- To actually create a copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and

Deep Copy

- Example:

```
int[] values = {7
int[] other = new
for (int i = 0; i <
    other[i] = values[i];
}
```

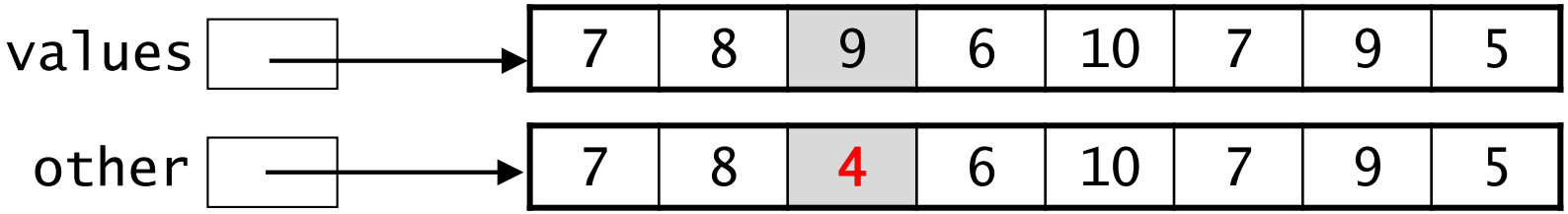


Copying an Array

- To actually create a copy of an array, we can:
  - create a new array of the same length as the first
  - traverse the arrays and copy the individual elements

- Example:

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
int[] other = new int[values.length];
for (int i = 0; i < values.length; i++) {
    other[i] = values[i];
}
```

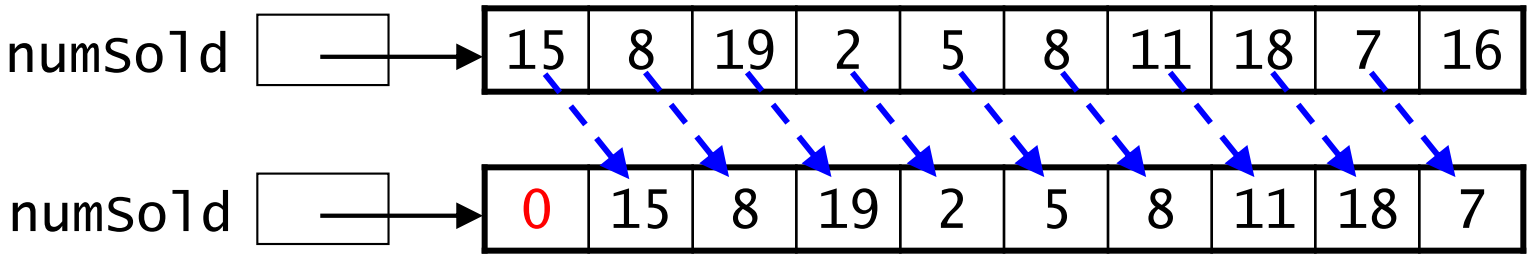


- What do the following lines print now?

```
other[2] = 4;
System.out.println(9 + " " + 4);
```

Shifting Values in an Array (cont.)

- At the start of each day, it's necessary to shift the values over to make room for the new day's sales.



  - the last value is lost, since it's now 10 days old

- In order to shift the values over, we need to perform assignments like the following:

```
numSold[9] = numSold[8];
numSold[6] = numSold[5];
numSold[2] = numSold[1];
```

  - what is the general form (the pattern) of these assignments?

```
numSold[i] = numSold[i - 1];
```

Shifting Values in an Array (cont.)

- Here's one attempt at code for shifting all of the elements:

```
for (int i = 0; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

- If we run this, we get an `ArrayIndexOutOfBoundsException`.

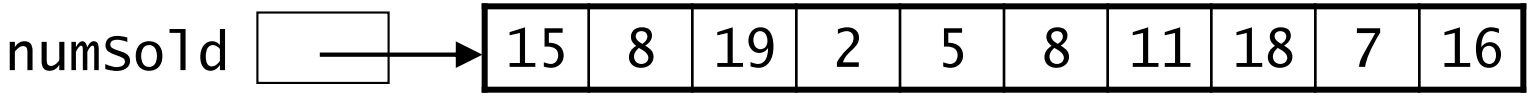
```
numSold[0] = numSold[-1];
```

Shifting Values in an Array (cont.)

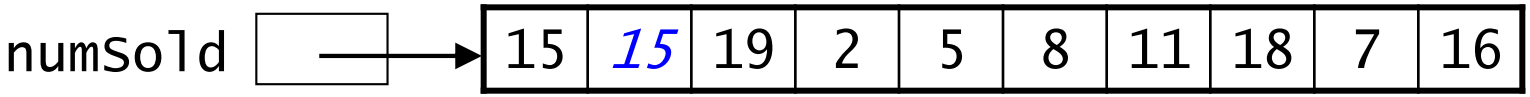
- This version of the code eliminates the exception:

```
for (int i = 1; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

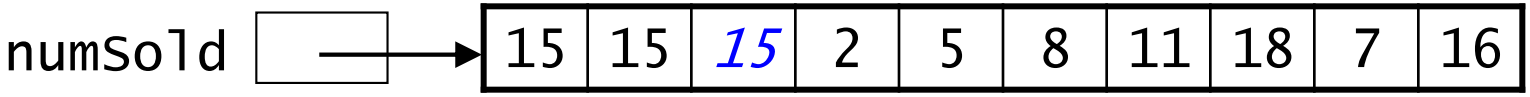
- Let's trace it to see what it does:



  - when `i == 1`, we perform `numSold[1] = numSold[0]` to get:



  - when `i == 2`, we perform **`numSold[2] = numSold[1]`** to get:



this obviously doesn't work!

Shifting Values in an Array (cont.)

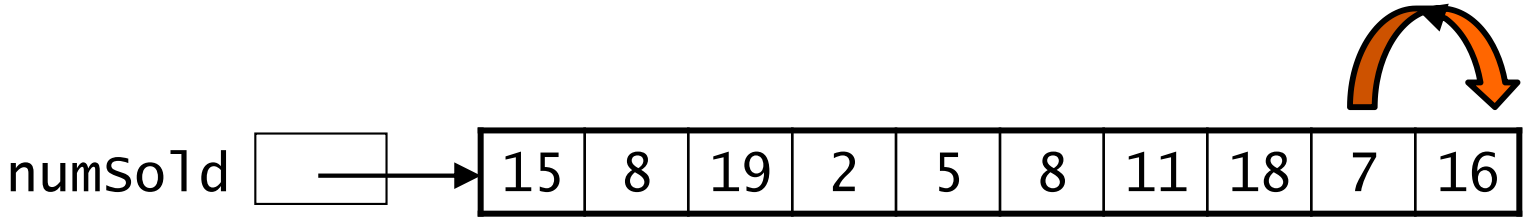
- How can we fix this code so that it does the right thing?

```
for (int i = 1; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

```
for (int i = numSold.length - 1; i > 0; i--) {

    numSold[i] = numSold[i - 1];

}
```



Shifting Values in an Array (cont.)

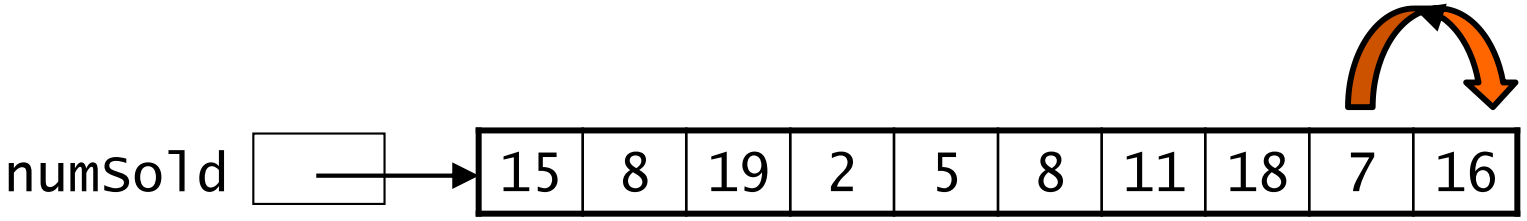
- How can we fix this code so that it does the right thing?

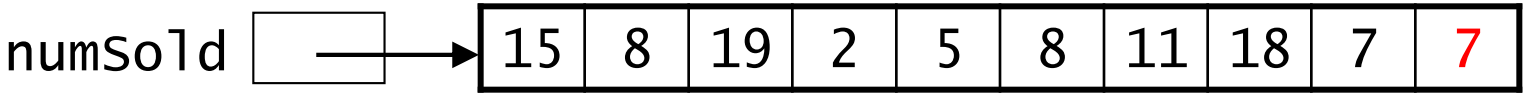
```
for (int i = 1; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

```
for (int i = numSold.length - 1; i > 0; i--) {

    numSold[i] = numSold[i - 1];

}
```





Shifting Values in an Array (cont.)

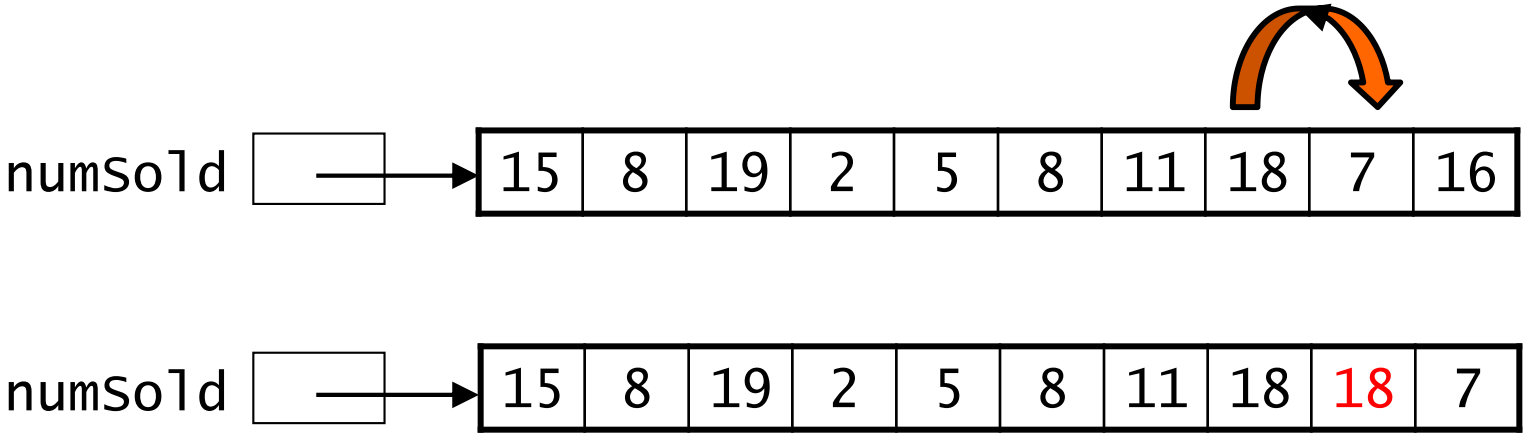
- How can we fix this code so that it does the right thing?

```
for (int i = 1; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

```
for (int i = numSold.length - 1; i > 0; i--) {

    numSold[i] = numSold[i - 1];

}
```



Shifting Values in an Array (cont.)

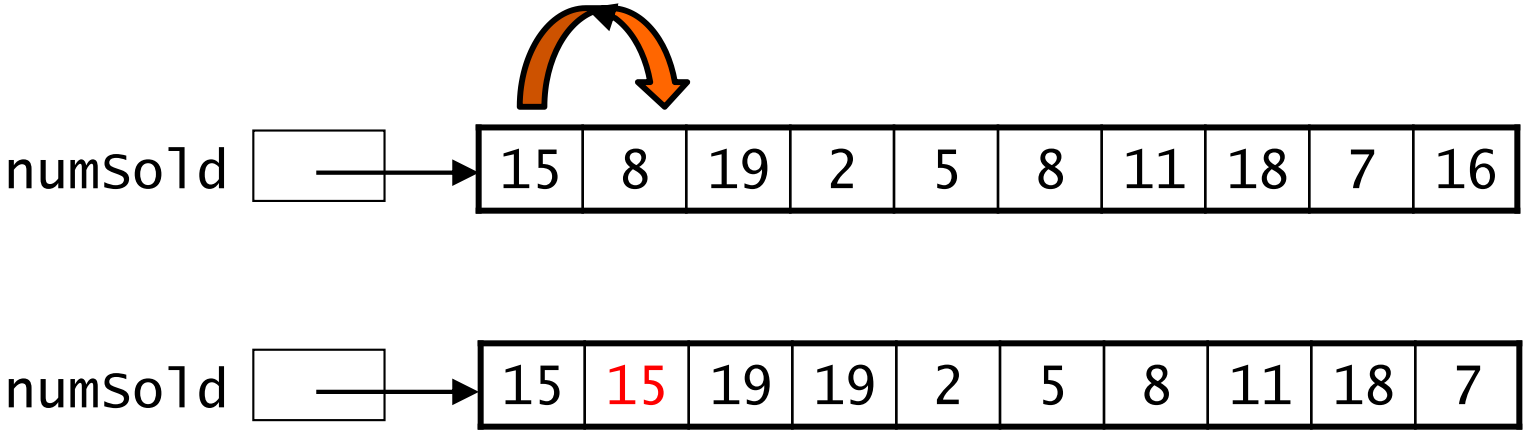
- How can we fix this code so that it does the right thing?

```
for (int i = 1; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

```
for (int i = numSold.length - 1; i > 0; i--) {

    numSold[i] = numSold[i - 1];

}
```



Shifting Values in an Array (cont.)

- How can we fix this code so that it does the right thing?

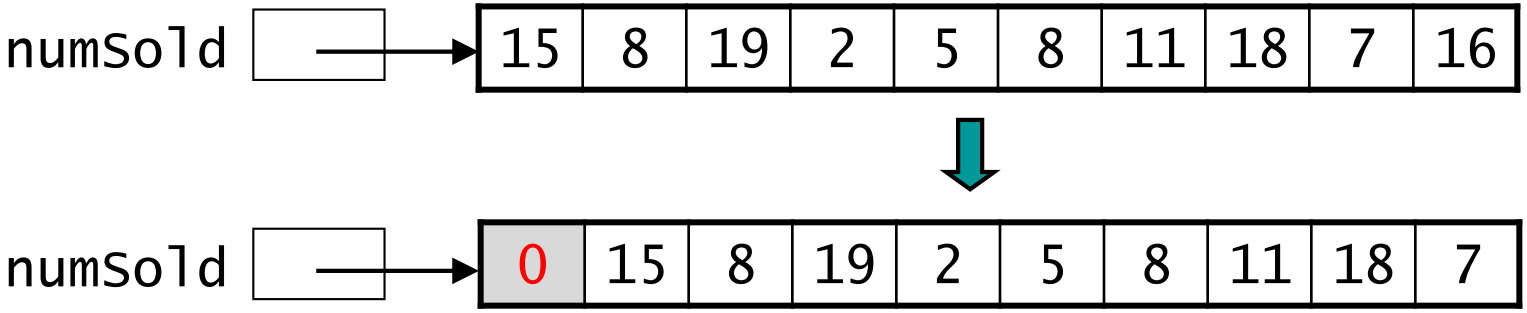
```
for (int i = 1; i < numSold.length; i++) {
    numSold[i] = numSold[i - 1];
}
```

```
for (int i = numSold.length - 1; i > 0; i--) {

    numSold[i] = numSold[i - 1];

}
```

- After performing all of the shifts, we would do: `numSold[0] = 0;`



"Growing" an Array

- Once we have created an array, we can't increase its size.

- Instead, we need to do the following:
  - create a new, larger array (use a temporary variable)
  - copy the contents of the original array into the new array
  - *assign the new array to the original array variable*

- Example for our `values` array:

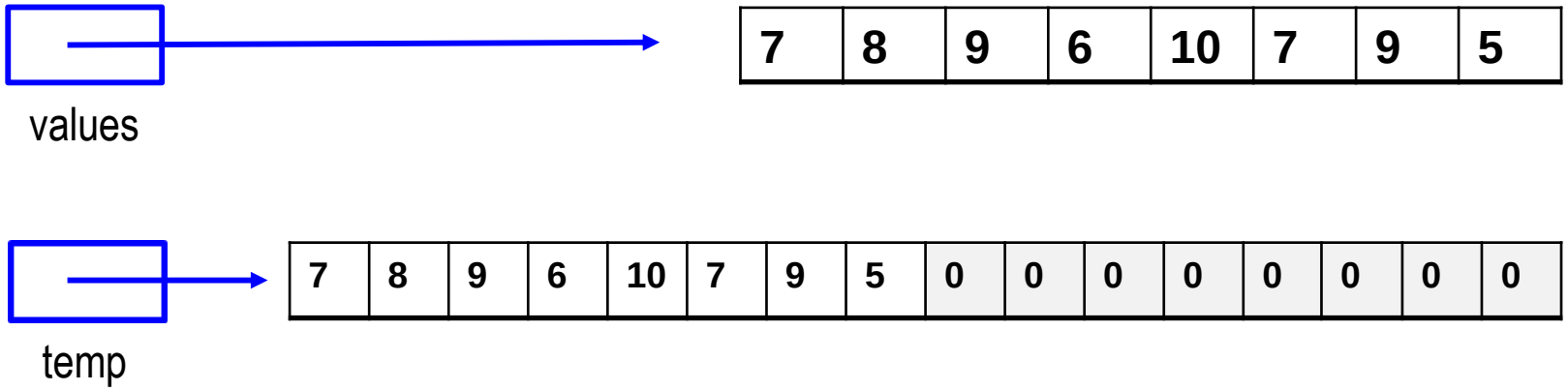
```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
...
int[] temp = new int[values.length*2];
for (int i = 0; i < values.length; i++) {
     temp[i] = values[i];
}
values = temp;
```

Example: Memory layout

Growing an Array

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
...
int[] temp = new int[values.length*2];
for (int i = 0; i < values.length; i++) {
     temp[i] = values[i];
}

// re-assign the temporary array
Values = temp;
```

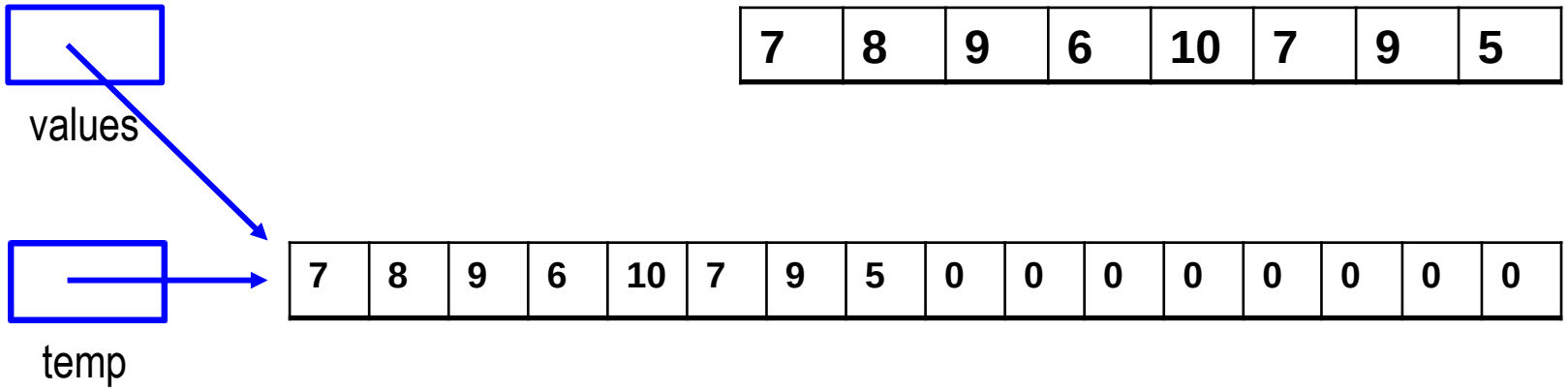


Example: Memory layout

Growing an Array

```
int[] values = {7, 8, 9, 6, 10, 7, 9, 5};
...
int[] temp = new int[values.length*2];
for (int i = 0; i < values.length; i++) {
     temp[i] = values[i];
}

// re-assign the temporary array
Values = temp;
```



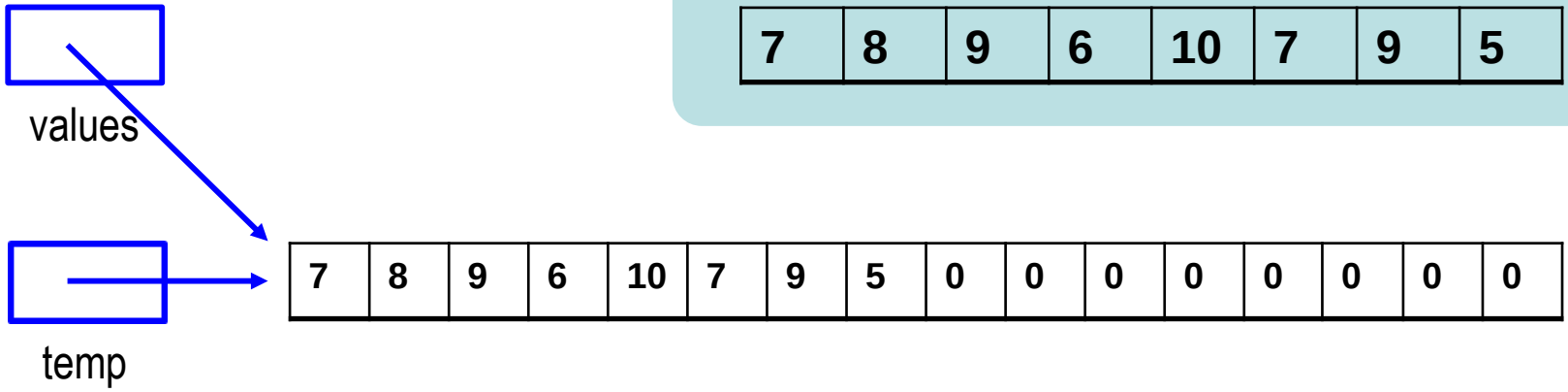
Example: Memory layout

Growing an Array

```
int[] values = {7, 8,
...
int[] temp = new in
for (int i = 0; i <
     temp[i] = value
}

// re-assign the temporary array
Values = temp;
```

What happens to the memory originally allocated?



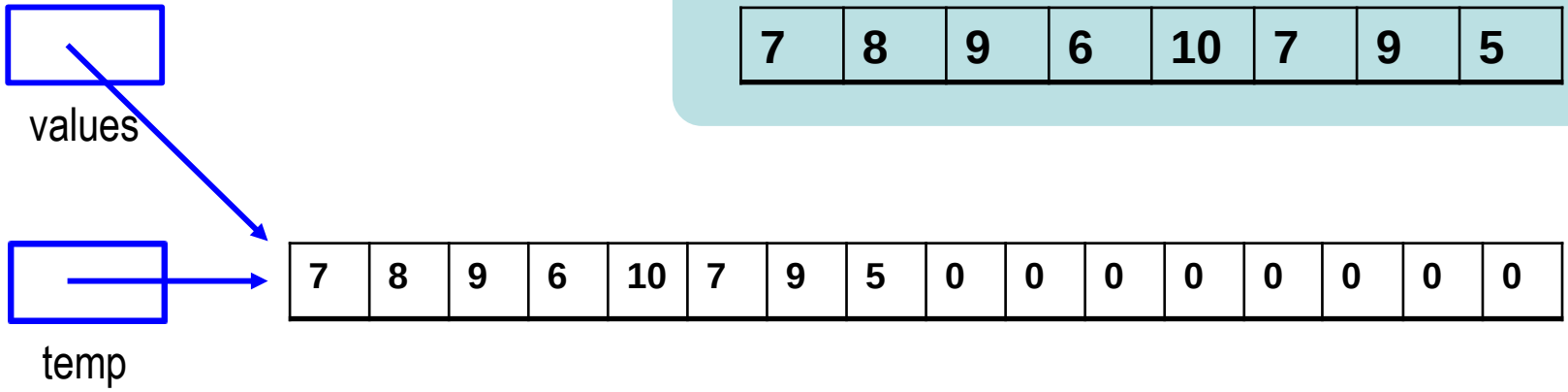
Example: Memory layout

Growing an Array

```
int[] values = {7, 8,
...
int[] temp = new in
for (int i = 0; i <
     temp[i] = value
}

// re-assign the temporary array
Values = temp;
```

memory remains "in limbo" waiting for the ***garbage collector*** *to free it!*



2-D Array Basics

- Example of declaring and creating a 2-D array:

```
int[][] matrix = new int[5][8];
```

number of rows

number of columns

- As in Python, we access an element by specifying 2 indices

  array[*row*][*column*]

  - example: `matrix[3][4]` gives the value at row 3, column 4

Example Application: Maintaining a Game Board

- For a Tic-Tac-Toe board, we could use a 2-D array to keep track of the state of the board:

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

- Alternatively, we could create *and* initialize it as follows:

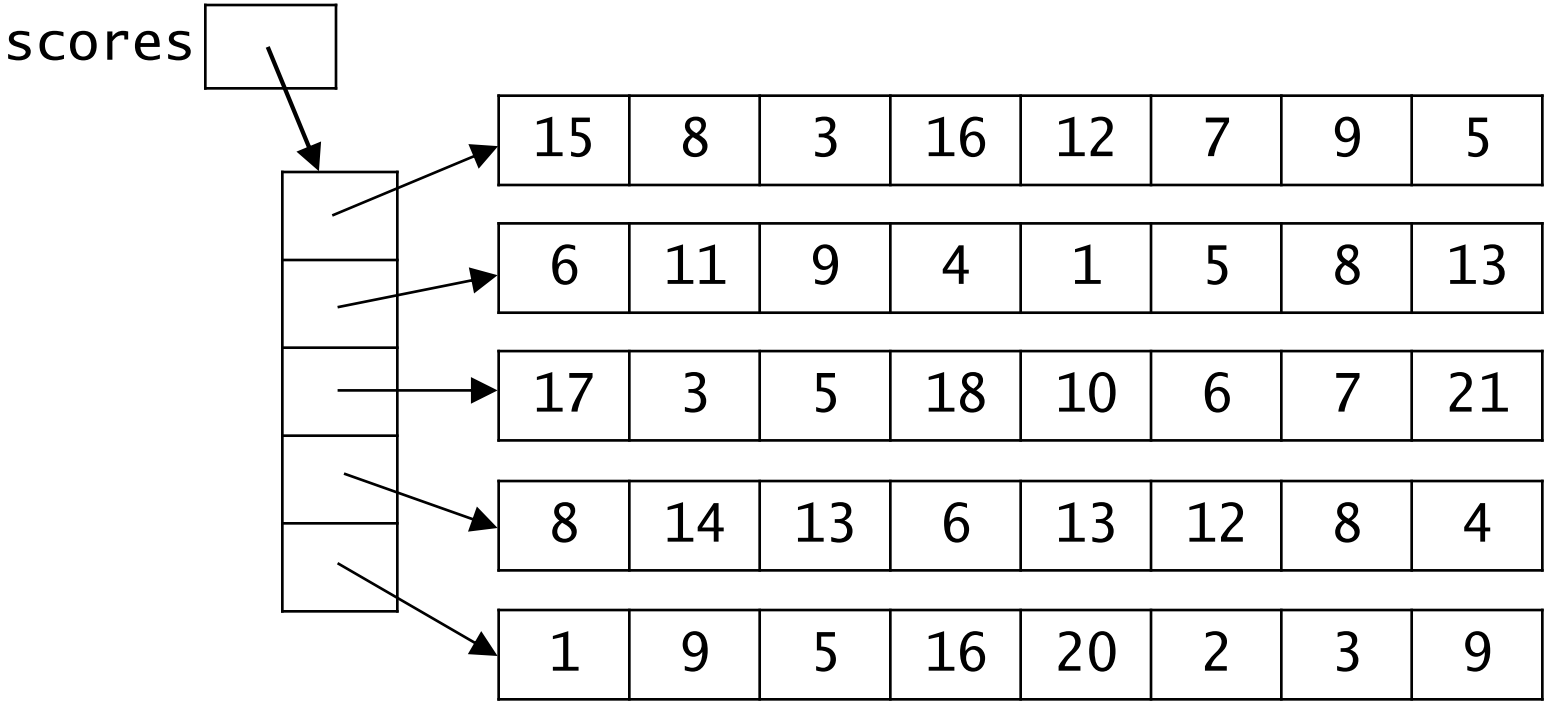
```
char[][] board = {{' ', ' ', ' '},
                  {' ', ' ', ' '},
                  {' ', ' ', ' '}};
```

- If a player puts an X in the middle square, we could record this fact by making the following assignment:

```
board[1][1] = 'X';
```

An Array of Arrays

- A 2-D array is really an array of arrays!

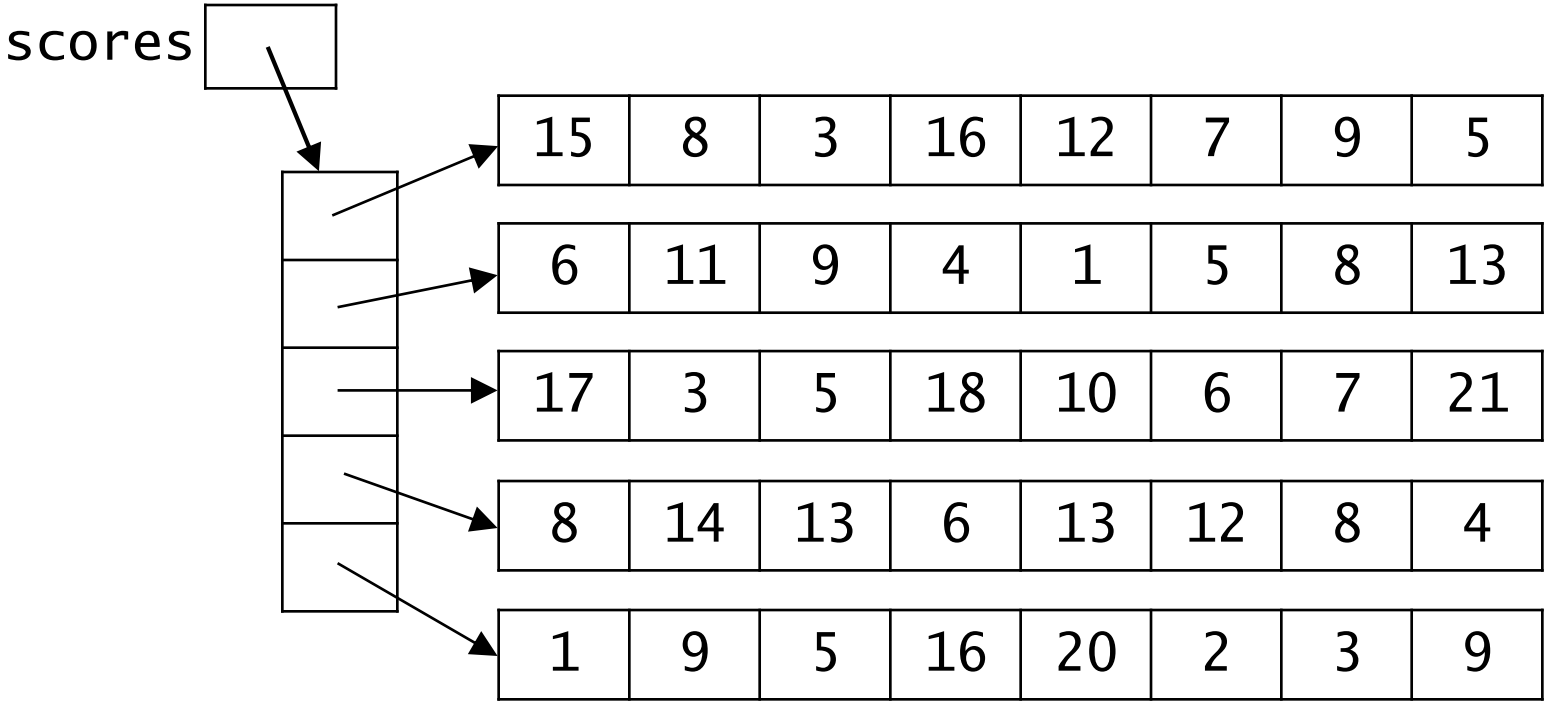


An Array of Arrays

- A 2-D array is really an array of arrays!

An Array of Arrays

- A 2-D array is really an array of arrays!



- `scores[0]` represents the entire first row
  `scores[1]` represents the entire second row, etc.

- *scores*`.length` gives the number of rows
  scores[row].length gives the number of columns in that row

Processing All of the Elements in a 2-D Array

- To perform some operation on all of the elements in a 2-D array, we typically use a nested loop.
  - example: finding the maximum value in a 2-D array.

```
public static int maxValue(int[][] arr) {
    int max = arr[0][0];
    for (int r = 0; r < arr.length; r++) {
        for (int c = 0; c < arr[r].length; c++) {
            if (arr[r][c] > max) {
                max = arr[r][c];
            }
        }
    }

    return max;
}
```

Processing All of the Elements in a 2-D Array

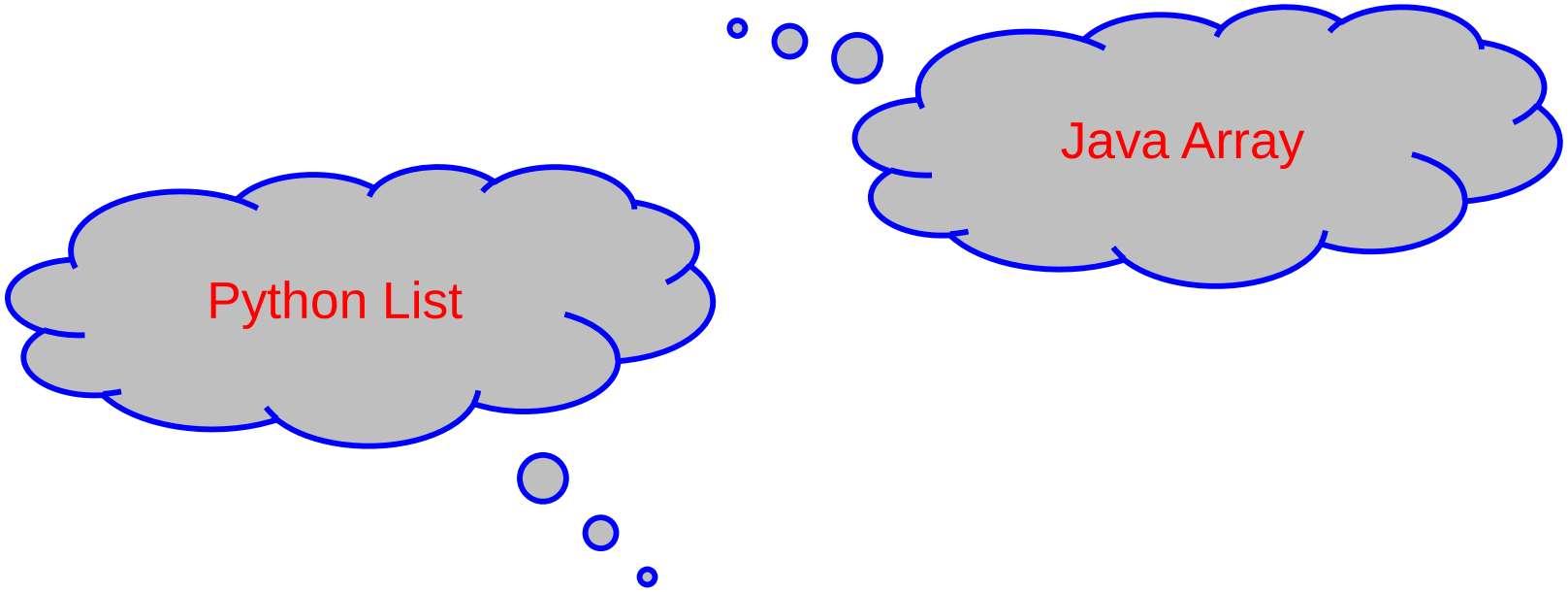
- To perform some operation on all of the elements in a 2-D array, we typically use a nested loop.
  - example: finding the maximum value in a 2-D array.

```
public static int maxValue(int[][] arr) {
    int max = arr[0][0];
    for (int r = 0; r < arr.length; r++) {
        for (int c = 0; c < arr[r].length; c++) {
            if (arr[r][c] > max) {
                max = arr[r][c];
            }
        }
    }

    return max;
}
```

Array vs. Python Lists

```
int arr = {51, 50, 36, 29, 30};
```




```
temps = [51, 50, 36, 29, 30]
```

Basic Operations on Lists vs. Arrays (cont.)

Python

```python
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```java
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

Basic Operations on Lists vs. Arrays (cont.)

Python

```
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

- Python uses [ ] to both:
  - surround list literals
  - index into the list

- Java uses:
  - { } to surround array literals
  - [ ] to index into the array

Basic Operations on Lists vs. Arrays (cont.)

Python

```python
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```java
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);
```

- Python uses `[ ]` to both:
  - surround list literals
  - index into the list
    - from both ends *(of the list)*

- Java uses:
  - `{ }` to surround array literals
  - `[ ]` to index into the array
    - cannot use negative indices

Basic Operations on Lists vs. Arrays (cont.)

Python

```
temps = [51, 50, 36, 29, 30]
first = temps[0]
num_temps = len(temps)
last = temps[-1]

temps[2] = 40
temps[3] += 5
print(temps[3])
print(temps)
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
int first = temps[0];
int numTemps = temps.length;
int last = temps[numTemps - 1];

temps[2] = 40;
temps[3] += 5;
System.out.println(temps[3]);
System.out.println(temps);  // no!
```

- `len(values)` gives the length of the list `values`
- printing a list displays its contents.

- `temps.length` gives the length of the array `values`
  - `length` is *not* a method, it is an attribute of the Arrays class
  - recall finding the length of a string: `s.length()`
- printing an array **does *not*** display its contents

Other Differences

Python

```
temps = [51, 50, 36, 29, 30]
first_two = temps[0:2]
temps = temps + [45, 29]
new_temps = [65] * 5
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
// no operator for slicing!
// no operator for concatenating!
// no operator for multiplying!
```

- In Java, the only array operator is `[]` for indexing.
- The `Array` class has static methods that provide the functionality of some of Python's operators.
  - example: `Arrays.copyOfRange(values, start, end)` returns the *slice* `values[start : end]`

Note the use of the *dot* operator to connect the method with the class!

Other Differences

Python

```
temps = [51, 50, 36, 29, 30]
first_two = temps[0:2]
temps = temps + [45, 29]
new_temps = [65] * 5
```

Java

```
int[] temps = {51, 50, 36, 29, 30};
// no operator for slicing!
// no operator for concatenating!
// no operator for multiplying!
```

- In Java, the only array operator is `[]` for indexing.
- The `Array` class has static methods that provide the functionality of some of Python's operators.
  - example: `Arrays.copyOfRange(values, start, end)` returns the slice `values[start : end]`
- If you really need the extra functionality, it's more common to use one of Java's built-in *collection classes*.
  - they allow you to construct a list *object* for a sequence
  - we'll soon be building our own collection classes!

Constructing an Array

Python

```
temps = [0] * 4
```

Java

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

- General pattern:

type`[]` *variable* `= new` *type*`[`*length*`];`

```
double[] vals = new double[100];    // array for 100 doubles
String[] names = new String[10];    // array for 10 string
                                    // references!
```

Constructing an Array

Python

```
temps = [0] * 4
```

Java

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

- General pattern:

type`[]` *variable* `= new` *type*`[`*length*`];`

```
double[] vals = new double[100];   // array for 100 doubles
String[] names = new String[10];   // array for 10 Strings
```

- Initially, the arrays are filled with the default value of their type:

`int`	`0`	`boolean`	`false`
`double`	`0.0`	objects	the special value `null`

Constructing and Filling a List / Array

Python

```
temps = [0] * 4
```

Java

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

Now that we have created
these structures,
how can we fill them
with data?

Constructing and Filling a List / Array

Python

```
temps = [0] * 4
print('enter 4 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[4];
System.out.println('enter 4 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

…. print out the contents of the array...

Constructing and Filling a List / Array

Python

```
temps = [0] * 100
print('enter 100 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

Anyone want to explicitly write 100 input statements?

Constructing and Filling a List / Array

Python

```
temps = [0] * 100
print('enter 100 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

more significantly, what if we do not know until run time the number of elements we need?

Constructing and Filling a List / Array

Python

```
temps = [0] * 100
print('enter 100 temps:')
temps[0] = int(input())
temps[1] = int(input())
temps[2] = int(input())
temps[3] = int(input())
print(temps)
```

Java

```
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
temps[0] = scan.nextInt();
temps[1] = scan.nextInt();
temps[2] = scan.nextInt();
temps[3] = scan.nextInt();
System.out.println(
  Arrays.toString(temps));
```

A more practical alternative is to use a loop!

Constructing and Filling a List / Array:

arrays and loops

Python

```python
temps = [0] * 100
print('enter 100 temps:')
for i in range(100):
    temps[i] = int(input())
print(temps)
```

Java

```java
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
for (int i = 0; i < 100; i++) {
    temps[i] = scan.nextInt();
}
System.out.println(
  Arrays.toString(temps));
```

To make the code more flexible…

Python

```python
temps = [0] * 100
print('enter 100 temps:')
for i in range(len(temps)):
    temps[i] = int(input())
print(temps)
```

Java

```java
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
for (int i = 0; i < temps.length; i++) {
    temps[i] = scan.nextInt();
}
System.out.println(
  Arrays.toString(temps));
```

Code to sum all the values of the array…

Python

```python
temps = [0] * 100
print('enter 100 temps:')
for i in range(len(temps)):
    temps[i] = int(input())
print(temps)
```

Java

```java
Scanner scan = new Scanner(System.in);
int[] temps = new int[100];
System.out.println('enter 100 temps:');
for (int i = 0; i < temps.length; i++) {
    temps[i] = scan.nextInt();
}
System.out.println(
  Arrays.toString(temps));
```

```java
int sum = 0;

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

This is important, because accessing an array *out of bounds* will result in a program exception.