

## Arrays

Computer Science S-111  
Harvard University

David G. Sullivan, Ph.D.

## Collections of Data

- Recall our program for averaging quiz grades:

```
public static void main(String[] args) {  
    Scanner console = new Scanner(System.in);  
    int total = 0;  
    int numGrades = 0;  
    while (true) {  
        System.out.print("Enter a grade (or -1 to quit): ");  
        int grade = console.nextInt();  
        if (grade == -1) {  
            break;  
        }  
        total += grade;  
        numGrades++;  
    }  
    if (numGrades > 0) {  
        ...  
    }  
}
```

- What if we wanted to store the individual grades?
  - an example of a *collection* of data

## Arrays

- An *array* is a collection of data values of the same type.
- In the same way that we think of a variable as a single box, an array can be thought of as a sequence of boxes:

0	1	2	3	4	5	6	7	← indices
7	8	9	6	10	7	9	5	← elements

- Each box contains one of the data values in the collection
  - referred to as the *elements* of the array
- Each element has a numeric *index*
  - the first element has an index of 0, the second element has an index of 1, etc.
  - example: the value 6 above has an index of 3
  - like the index of a character in a String

## Declaring and Creating an Array

- We use a variable to represent the array as a whole.
- Example of declaring an array variable:

```
int[] grades;
```

- the `[]` indicates that it will represent an array
- the `int` indicates that the elements will be `ints`

- Declaring the array variable does *not* create the array.
- Example of creating an array:

```
grades = new int[8];
```

↑  
the *length* of the array –  
i.e., the number of elements

## Declaring and Creating an Array (cont.)

- We often declare and create an array in the same statement:

```
int[] grades = new int[8];
```

- General syntax:

```
<type>[] <array> = new <type>[<length>];
```

where

<type> is the type of the individual elements

<array> is the name of the variable used for the array

<length> is the number of elements in the array

## The Length of an Array

- The *length* of an array is the number of elements in the array.
- The length of an array can be obtained as follows:

```
<array>.length
```

- example:

```
grades.length
```

- note: it is *not* a method

```
grades.length() won't work!
```

## Auto-Initialization

- When you create an array in this way:

```
int[] grades = new int[8];
```

the runtime system gives the elements default values:

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

- The value used depends on the type of the elements:

int	0
double	0.0
char	'\0'
boolean	false
objects	null

## Accessing an Array Element

- To access an array element, we use an expression of the form

```
<array>[<index>]
```

- Examples:

```
grades[0] accesses the first element
grades[1] accesses the second element
grades[5] accesses the sixth element
```

- Here's one way of setting up the array we showed earlier:

0	1	2	3	4	5	6	7
7	8	9	6	10	7	9	5

```
int[] grades = new int[8];
grades[0] = 7; grades[1] = 8; grades[2] = 9;
grades[3] = 6; grades[4] = 10; grades[5] = 7;
grades[6] = 9; grades[7] = 5;
```

## Accessing an Array Element (cont.)

- Acceptable index values:  
integers from 0 to `<array>.length - 1`
- If we specify an index outside that range, we'll get an `ArrayIndexOutOfBoundsException` at runtime.

- example:

```
int[] grades = int[8];  
grades[8] = 5;
```

0	1	2	3	4	5	6	7	8
0	0	0	0	0	0	0	0	<i>no such element!</i>

## Accessing an Array Element (cont.)

- The index can be any integer expression.
  - example:  

```
int lastGrade = grades[grades.length - 1];
```
- We can operate on an array element in the same way that we operate on any other variable of that type.
  - example: applying a 10% late penalty to the grade at index `i`  

```
grades[i] = (int)(grades[i] * 0.9);
```
  - example: adding 5 points of extra credit to the grade at index `i`  

```
grades[i] += 5;
```

## Another Way to Create an Array

- If we know that we want an array to contain specific values, we can specify them when create the array.
- Example: here's another way to create and initialize our grades array:

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

- The list of values is known as an *initialization list*.
  - it can only be specified when the array is declared
  - we don't use the new operator in this case
  - we don't specify the length of the array – it is determined from the number of values in the initialization list
- Other examples:

```
double[] heights = {65.2, 72.0, 70.6, 67.9};  
boolean[] isPassing = {true, true, false, true};
```

## Storing Grades Entered by the User

- We need to know how big to make the array.
  - one way: ask the user for the maximum number of values

```
public static void main(String[] args) {  
    Scanner console = new Scanner(System.in);  
  
    System.out.print("How many grades? ");  
    int maxNumGrades = console.nextInt();  
    int[] grades = new int[maxNumGrades];  
  
    int total = 0;  
    int numGrades = 0;  
  
    while (numGrades < maxNumGrades) {  
        System.out.print("Enter a grade (or -1 to quit): ");  
        grades[numGrades] = console.nextInt();  
        if (grades[numGrades] == -1) {  
            break;  
        }  
        total += grades[numGrades];  
        numGrades++;  
    }  
    ...  
}
```

## Processing the Values in an Array

- We often use a for loop to process the values in an array.

- Example: print out all of the grades

```
int[] grades = new int[maxNumGrades];  
...  
for (int i = 0; i < grades.length; i++) {  
    System.out.println("grade " + i + ": " + grades[i]);  
}
```

- General pattern:

```
for (int i = 0; i < <array>.length; i++) {  
    do something with <array>[i];  
}
```

- Processing array elements sequentially from first to last is known as *traversing* the array.
  - noun = *traversal*

## Another Example of Traversing an Array

- Let's write code to find the highest quiz grade in the array:

```
int max = _____;  
for (_____; _____; _____) {  
  
    }  
}
```

## Another Example of Traversing an Array (cont.)

grades array: 

7	8	9	6	10	7	9	5
---	---	---	---	----	---	---	---

- Let's trace through our code:  

```
int max = grades[0];  
for (int i = 1; i < grades.length; i++) {  
    if (grades[i] > max) {  
        max = grades[i];  
    }  
}
```

<u>i</u>	<u>grades[i]</u>	<u>max</u>
		7
1	8	8
2	9	9
3	6	9
4	10	10
5	7	10
...		

## Review: What Is a Variable?

- We've seen that a variable is like a named "box" in memory that can be used to store a value.

`int count = 10;`                      count 

10
----

- If a variable represents a primitive-type value, the value is stored in the variable itself, as shown above.



## Reference Variables

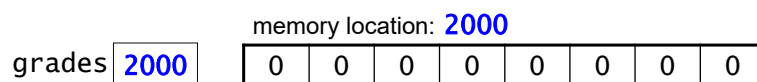
- If a variable represents an object, the object itself is *not* stored inside the variable.
- Rather, the object is located somewhere else in memory, and the variable holds the *memory address* of the object.
  - we say that the variable stores a *reference* to the object
  - such variables are called *reference variables*

## Arrays and References

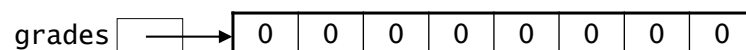
- An array is a type of object.
- Thus, an array variable is a reference variable.
  - it stores a reference to the array
- Example:

```
int[] grades = new int[8];
```

might give the following picture:



- We usually use an arrow to represent a reference:



## Printing an Array

- What is the output of the following lines?  

```
int[] grades = {7, 8, 9, 6, 10, 7, 9, 5};  
System.out.println(grades);
```
- To print the contents of the array, we can use a for loop as we showed earlier.
- We can also use the `Arrays.toString()` method, which is part of Java's built in `Arrays` class.  

```
int[] grades = {7, 8, 9, 6, 10, 7, 9, 5};  
System.out.println(Arrays.toString(grades));
```

  - doing so produces the following output:  
`[7, 8, 9, 6, 10, 7, 9, 5]`
- To use this method, we need to import the `java.util` package.

## What is the output of the full program?

```
import java.util.*;  
public class FunWithArrays {  
    public static void main(String[] args) {  
        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(Arrays.toString(temps));  
    }  
}
```

temps   
first   
numTemps   
last

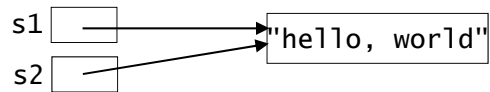
output:



## Copying References

- When we assign the value of one reference variable to another, we copy the reference to the object. We do *not* copy the object itself.
- Example involving objects:

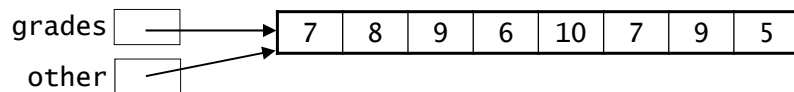
```
String s1 = "hello, world";  
String s2 = s1;
```



## Copying References (cont.)

- An example involving an array:

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



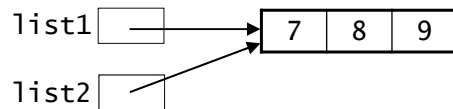
- Given the lines of code above, what will the lines below print?  

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

## Changing the Internals vs. Changing a Variable

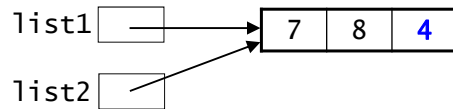
- When two variables hold a reference to the same array...

```
int[] list1 = {7, 8, 9};  
int[] list2 = list1;
```



- ...if we change *the internals* of the array, both variables will "see" the change:

```
list2[2] = 4;  
System.out.println(Arrays.toString(list1));
```

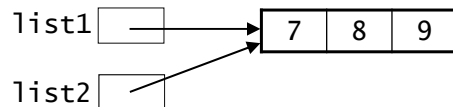


output of println:

## Changing the Internals vs. Changing a Variable (cont.)

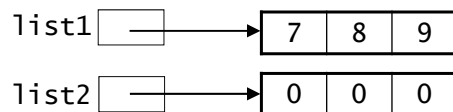
- When two variables hold a reference to the same array...

```
int[] list1 = {7, 8, 9};  
int[] list2 = list1;
```



- ...if we change one of the variables *itself*, that does *not* change the other variable:

```
list2 = new int[3];  
System.out.println(Arrays.toString(list1));
```



output of println:

## Null References

- To indicate that a reference variable doesn't yet refer to any object, we can assign it a special value called `null`.

```
int[] grades = null;  
String s = null;
```

grades null                      s null

- Attempting to use a null reference to access an object produces a `NullPointerException`.

- "pointer" is another name for reference

- examples:

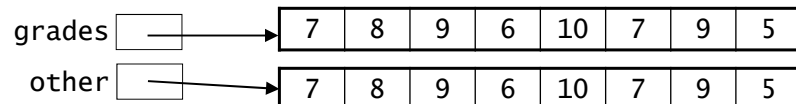
```
int[] grades = null;  
String s = null;  
grades[3] = 10;                      // NullPointerException!  
char ch = s.charAt(5);              // NullPointerException!
```

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



- What do the following lines print now?

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

## Programming Style Point

- Here's how we copied the array:

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

- This would also work:

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

- Why is the first way better?

## Passing an Array to a Method

- Let's put our code for finding the highest grade into a method:

```
public class GradeAnalyzer {
    public static _____ maxGrade(int[] grades) {
        int max = grades[0];
        for (int i = 1; i < grades.length; i++) {
            if (grades[i] > max) {
                max = grades[i];
            }
        }
        _____;
    }

    public static void main(String[] args) {
        ...
        int maxNumGrades = console.nextInt();
        int[] grades = new int[maxNumGrades];
        ... // code to read in the values
        System.out.println("max grade = " +
            _____);
    }
}
```

## Passing an Array to a Method (cont.)

- What's wrong with this alternative approach?

```
public class GradeAnalyzer {
    public static int maxGrade(int[] grades) {
        int max = grades[0];
        for (int i = 1; i < grades.length; i++) {
            if (grades[i] > max) {
                max = grades[i];
            }
        }

        return max;
    }

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

        int maxNumGrades = console.nextInt();
        int[] grades = new int[maxNumGrades];

        ... // code to read in the values

        maxGrade(grades);
        System.out.println("max grade = " + max);
    }
}
```

## Passing an Array to a Method (cont.)

- We could do this instead:

```
public class GradeAnalyzer {
    public static int maxGrade(int[] grades) {
        int max = grades[0];
        for (int i = 1; i < grades.length; i++) {
            if (grades[i] > max) {
                max = grades[i];
            }
        }

        return max;
    }

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

        int maxNumGrades = console.nextInt();
        int[] grades = new int[maxNumGrades];

        ... // code to read in the values

        int max = maxGrade(grades);
        System.out.println("max grade = " + max);
    }
}
```

## Finding the Average Value in an Array

- Here's a method that computes the average grade:

```
public static double averageGrade(int[] grades) {  
    int total = 0;  
    for (int i = 0; i < grades.length; i++) {  
        total += grades[i];  
    }  
    return (double)total / grades.length;  
}
```

## Testing If An Array Meets Some Condition

- Let's say that we need to be able to determine if there are any grades below a certain cutoff value.
  - e.g., to determine if a retest should be given
- Does this method work?

```
public static boolean  
anyGradesBelow(int[] grades, int cutoff) {  
    for (int i = 0; i < grades.length; i++) {  
        if (grades[i] < cutoff) {  
            return true;  
        } else {  
            return false;  
        }  
    }  
}
```



### Testing If An Array Meets Some Condition (cont.)

- We can return true as soon as we find a grade that is below the threshold.
- We can only return false if *none* of the grades is below.
- Here is a corrected version:

```
public static boolean
anyGradesBelow(int[] grades, int cutoff) {
    for (int i = 0; i < grades.length; i++) {
        if (grades[i] < cutoff) {
            return true;
        }
    }

    // if we get here, none of the grades is below.
    return false;
}
```

### Testing If An Array Meets Some Condition (cont.)

- Here's a similar problem: write a method that determines if all of the grades are perfect (assume perfect = 100).

```
public static boolean allPerfect(int[] grades) {

}

}
```

## Using an Array to Count Things

- Let's say that we want to count how many times each of the possible grade values appears in a collection of grades.
- We can use an array to store the counts.
  - `counts[i]` will store the number of times that the grade `i` appears
  - for this grades array

grades 

7	8	9	6	10	7	9	5
---	---	---	---	----	---	---	---

we would have this array of counts:

counts 

0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	1	1	2	1	2	1

## Using an Array to Count Things (cont.)

grades 

7	8	9	6	10	7	9	5
---	---	---	---	----	---	---	---

counts 

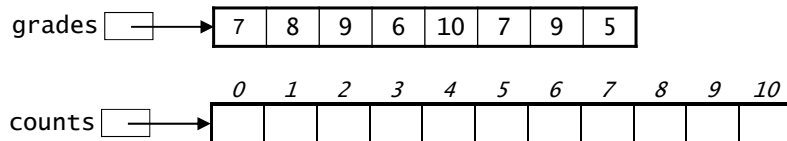
0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	1	1	2	1	2	1

- The size of the counts array should be one more than the maximum value being counted:

```
int max = maxGrade(grades);
int[] counts = new int[max + 1];
```
- Given the array, here's how to do the actual counting:

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

## Using an Array to Count Things (cont.)



- Let's trace through this code for the grades array shown above:

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

<u>i</u>	<u>grades[i]</u>	<u>operation performed</u>
----------	------------------	----------------------------

## A Method That Returns an Array

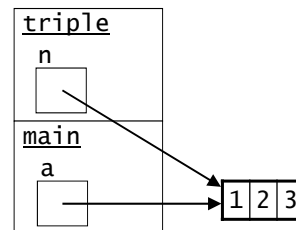
- We can write a method to create and return the array of counts:

```
public static int[] getCounts(int[] grades, int maxGrade) {  
    int[] counts = new int[maxGrade + 1];  
    for (int i = 0; i < grades.length; i++) {  
        counts[grades[i]]++;  
    }  
    return counts;  
}  
  
public static void main(String[] args) {  
    ... // main method begins as in the earlier versions  
    int max = maxGrade(grades);  
    int[] counts = getCounts(grades, max);  
    ...  
}
```

## Using a Method to Change an Array's Contents

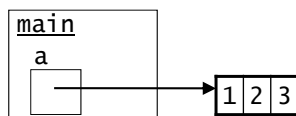
```
public static void main(String[] args) {  
    int[] a = {1, 2, 3};  
    triple(a);  
    System.out.println(Arrays.toString(a));  
}  
  
public static void triple(int[] n) {  
    for (int i = 0; i < n.length; i++) {  
        n[i] = n[i] * 3;  
    }  
}
```

- When a method is passed an array as a parameter, it gets a copy of the reference, *not* a copy of the array.
- If the method changes the internals of the array, those changes will be there after the method returns.

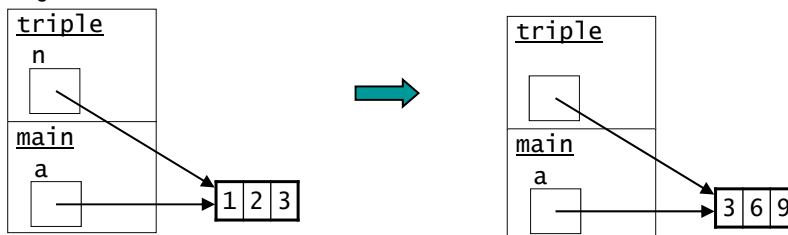


## Using a Method to Change an Array's Contents (cont.)

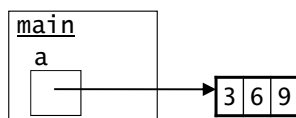
*before method call*



*during method call*



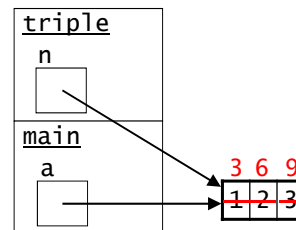
*after method call*



## Changing the Internals vs. Changing a Variable

```
public static void main(String[] args) {  
    int[] a = {1, 2, 3};  
    triple(a);  
    System.out.println(Arrays.toString(a));  
}  
  
public static void triple(int[] n) {  
    for (int i = 0; i < n.length; i++) {  
        n[i] = n[i] * 3;    // changes internals  
    }  
}
```

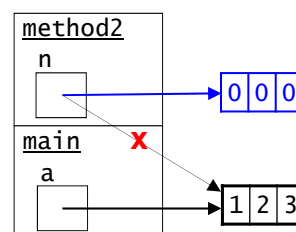
- If the method changes the *internals* of the array, those changes will be there after the method returns.



## Changing the Internals vs. Changing a Variable (cont.)

```
public static void main(String[] args) {  
    int[] a = {1, 2, 3};  
    triple(a);  
    System.out.println(Arrays.toString(a));  
}  
  
public static void method2(int[] n) {  
    n = new int[3];    // changes the variable  
}
```

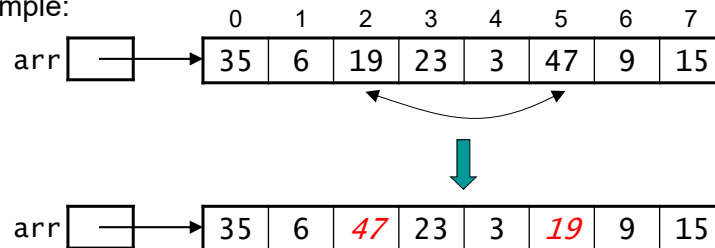
- However, if the method changes its *variable* for the array, that change does *not* affect the original array.
- Changing what's in one variable doesn't affect any other variable!



## Swapping Elements in an Array

- We sometimes need to be able to swap two elements in an array.

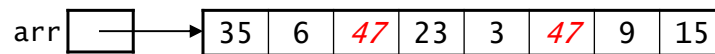
- Example:



- What's wrong with this code for swapping the two values?

```
arr[2] = arr[5];  
arr[5] = arr[2];
```

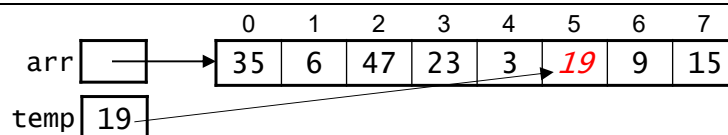
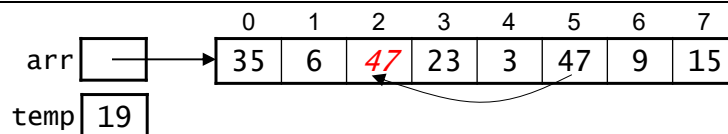
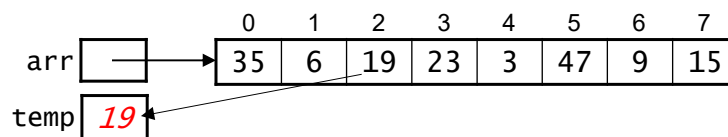
- it gives this:



## Swapping Elements in an Array (cont.)

- To perform a swap, we need to use a temporary variable:

```
int temp = arr[2];  
arr[2] = arr[5];  
arr[5] = temp;
```



## A Method for Swapping Elements

- Here's a method for swapping the elements at positions *i* and *j* in the array *arr*:

```
public static void swap(int[] arr, int i, int j) {  
    int temp = arr[i];  
    arr[i] = arr[j];  
    arr[j] = temp;  
}
```
- We don't need to return anything, because the method changes the internals of the array that is passed in.
- Here's an example of how we would use it:

```
int[] grades = {7, 8, 9, 6, 10, 7, 9, 5};  
swap(grades, 2, 5);  
System.out.println(Arrays.toString(grades));
```
- What would the output be?

## Recall: A Method That Returns an Array

- We can write a method to create and return the array of counts:

```
public static int[] getCounts(int[] grades, int maxGrade) {  
    int[] counts = new int[maxGrade + 1];  
    for (int i = 0; i < grades.length; i++) {  
        counts[grades[i]]++;  
    }  
    return counts;  
}
```

```
public static void main(String[] args) {  
    ... // main method begins as in the earlier versions  
    int max = maxGrade(grades);  
    int[] counts = getCounts(grades, max);  
    ...  
}
```

## An Alternative Approach for the Array of Counts

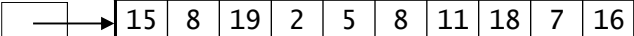
- Create the array ahead of time and pass it into the method:

```
public static void getCounts(int[] grades, int[] counts) {  
    for (int i = 0; i < grades.length; i++) {  
        counts[grades[i]]++;  
    }  
}  
  
public static void main(String[] args) {  
    ... // main method begins as in the earlier versions  
    int max = maxGrade(grades);  
    int[] counts = new int[max];  
    getCounts(grades, counts);  
    ...  
}
```

- Because the method changes the internals of the array, those changes will be there after the method returns.

## Shifting Values in an Array

- Let's say a small business is using an array to store the number of items sold over a 10-day period.

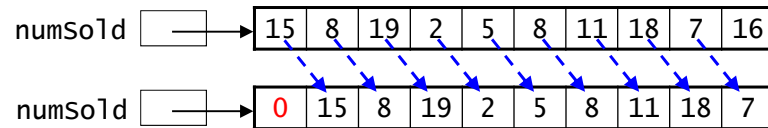
numSold 

numSold[0] gives the number of items sold today  
numSold[1] gives the number of items sold 1 day ago  
numSold[2] gives the number of items sold 2 days ago  
...  
numSold[9] gives the number of items sold 9 days ago



### 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[8] = numSold[7];`
  - `numSold[7] = numSold[6];`
  - `numSold[6] = numSold[5];`
  - `numSold[5] = numSold[4];`
  - `numSold[4] = numSold[3];`
  - `numSold[3] = numSold[2];`
  - `numSold[2] = numSold[1];`
  - `numSold[1] = numSold[0];`
- what is the general form (the pattern) of these assignments?

### 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`. Why?

## 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:

numSold 


15	8	19	2	5	8	11	18	7	16
----	---	----	---	---	---	----	----	---	----

- when  $i == 1$ , we perform  $\text{numSold}[1] = \text{numSold}[0]$  to get:

numSold 

15	15	19	2	5	8	11	18	7	16
----	----	----	---	---	---	----	----	---	----

- when  $i == 2$ , we perform  $\text{numSold}[2] = \text{numSold}[1]$  to get:

numSold 

15	15	15	2	5	8	11	18	7	16
----	----	----	---	---	---	----	----	---	----

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 ( ; ; ) {  
  
}
```

- After performing all of the shifts, we would do:  $\text{numSold}[0] = 0$ ;

numSold 

15	15	8	19	2	5	8	11	18	7
----	----	---	----	---	---	---	----	----	---



numSold 

0	15	8	19	2	5	8	11	18	7
---	----	---	----	---	---	---	----	----	---

## "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

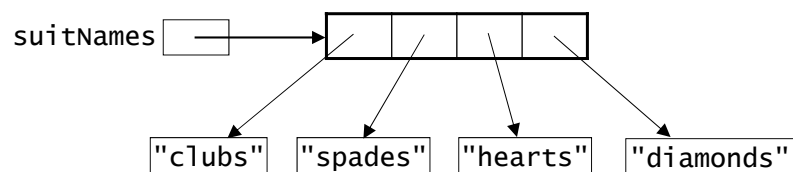
- Example for our grades array:

```
int[] grades = {7, 8, 9, 6, 10, 7, 9, 5};  
...  
int[] temp = new int[16];  
for (int i = 0; i < grades.length; i++) {  
    temp[i] = grades[i];  
}  
grades = temp;
```

## Arrays of Objects

- We can use an array to represent a collection of objects.
- In such cases, the cells of the array store references to the objects.
- Example:

```
String[] suitNames = {"clubs", "spades",  
    "hearts", "diamonds"};
```



## Two-Dimensional Arrays

- Thus far, we've been looking at single-dimensional arrays
- We can also create *multi-dimensional* arrays.
- The most common type is a two-dimensional (2-D) array.
- We can visualize it as a matrix consisting of rows and columns:

	0	1	2	3	4	5	6	7	← column indices
0	15	8	3	16	12	7	9	5	
1	6	11	9	4	1	5	8	13	
2	17	3	5	18	10	6	7	21	
3	8	14	13	6	13	12	8	4	
4	1	9	5	16	20	2	3	9	

row indices

## 2-D Array Basics

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

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

number  
of rows

number  
of columns

- To access an element, we use an expression of the form

`<array>[<row>][<column>]`

- example: `scores[3][4]` gives the score at row 3, column 4

	0	1	2	3	4	5	6	7
0	15	8	3	16	12	7	9	5
1	6	11	9	4	1	5	8	13
2	17	3	5	18	10	6	7	21
3	8	14	13	6	13	12	8	4
4	1	9	5	16	20	2	3	9

## 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:

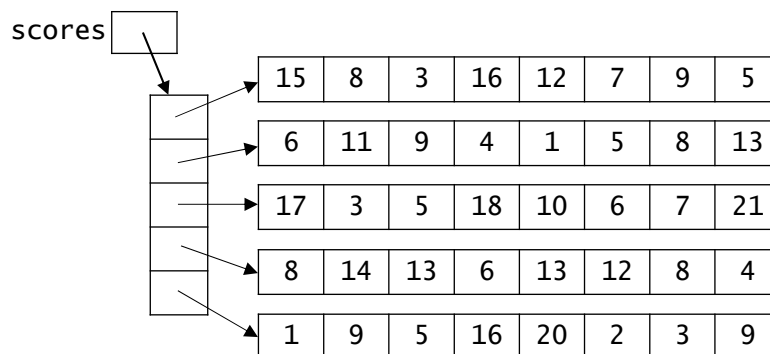
```
char[][] board = { {' ', ' ', ' ', ' ', ' '},  
                  {' ', ' ', ' ', ' ', ' '},  
                  {' ', ' ', ' ', ' ', ' '}};
```

- 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!



- scores[0] represents the entire first row  
scores[1] represents the entire second row, etc.
- <array>.length gives the number of rows  
<array>[<row>].length gives the number of columns in that row

## Processing All of the Elements in a 2-D Array

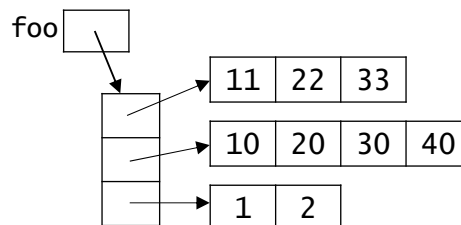
- To perform some operation on all of the elements in a 2-D array, we typically use a nested loop.
- example: finding the maximum value in a 2-D array.

```
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;  
}
```

## Optional: Other Multi-Dimensional Arrays

- It's possible to have a "ragged" 2-D array in which different rows have different numbers of columns:

```
int[][] foo = {{11, 22, 33},  
              {7, 20, 30, 40},  
              {1, 2}};
```



- We can also create arrays of higher dimensions.
- example: a three-dimensional matrix:  

```
double[][][] matrix = new double[2][5][4];
```