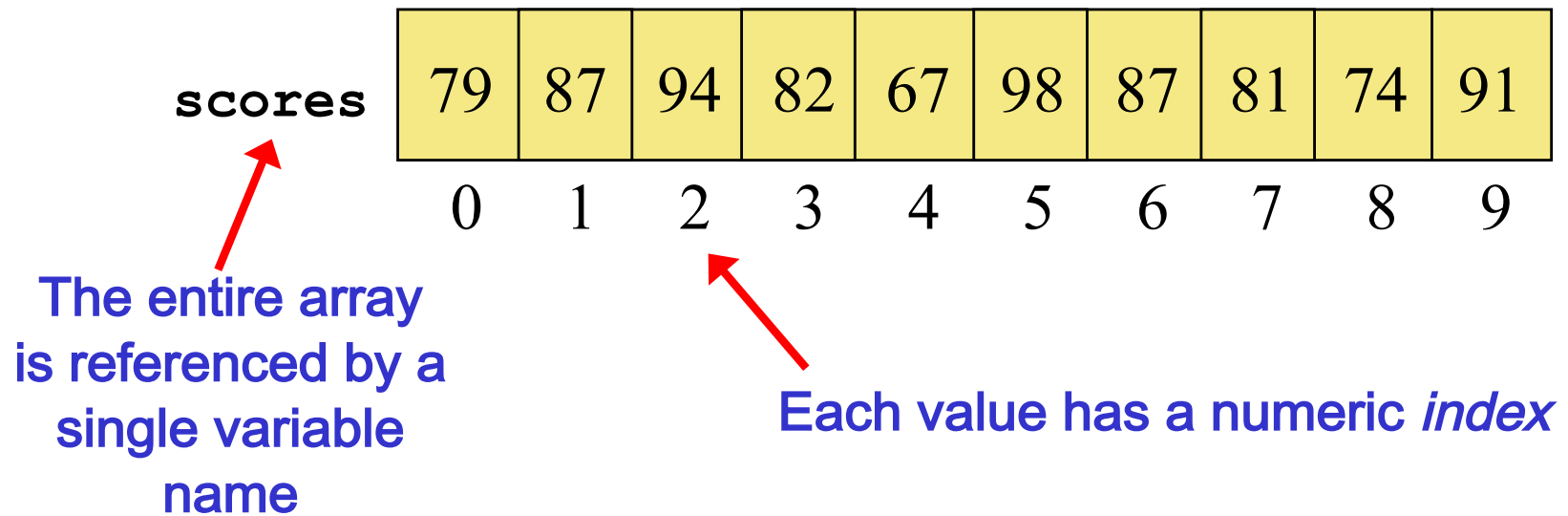


# Arrays

- Objectives - when we have completed this set of notes, you should be familiar with:
  - array declaration and use
  - bounds checking and capacity
  - arrays that store object references
  - command-line arguments
  - variable length parameter lists
  - multidimensional arrays

# Arrays

- An *array* is a container object that holds a fixed number of values of a single type.



An array of length **n** is indexed from **0** to **n-1**

This array holds **10** values that are indexed from **0** to **9**

# Declaring Arrays

- The `scores` array could be declared as follows:

```
int[] scores = new int[10];
```

- The type of the variable `scores` is `int[]` (an array of `int` or an **int array**); when you see `[]`, think or say array
  - The reference variable `scores` is set to a new array object that holds 10 values of type `int`; note the use of the **new** operator with the *type[length]*
- The array is part of the Java language (whereas `ArrayList` is a class in the Java class libraries as described in the Java API)

# Alternate Array Syntax

- The brackets of the array type can be associated with the element type or with the name of the array:

```
float [] prices;
```

```
float prices [];
```

- The first format generally is more readable and should be used
- **Remember** - Whenever you see [] brackets (a.k.a., square brackets) in Java, think or say array!

# Arrays

- The values held in an array are called array *elements*
  - The *element type* can be a primitive type or a reference type
- The declaration of an array variable does not create the array object; but rather only a variable that can reference the array

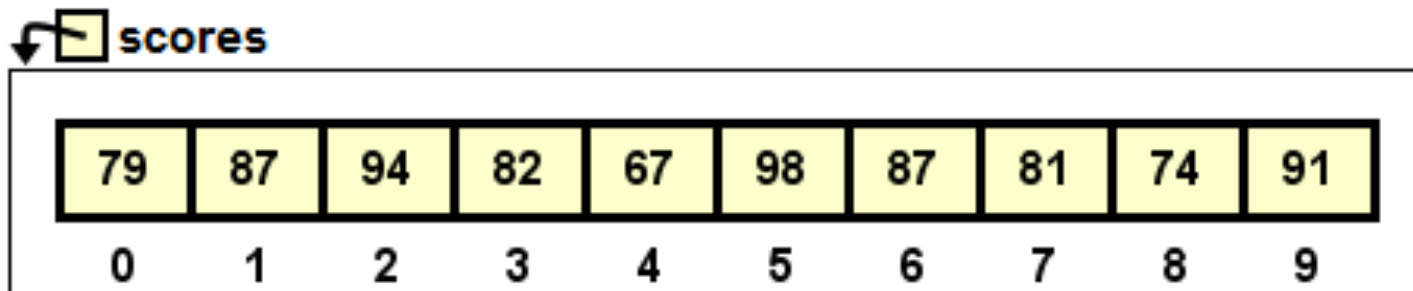
```
char[] letters;
```

- The **new** operator creates (or instantiates) the array with the specified number of elements

```
letters = new char[5];
```

# Accessing Array Elements

- Elements are accessed using the array name followed by the index in brackets
- The expression `scores[2]` evaluates to the value 94



Examples:

```
int singleScore = scores[2];
```

```
System.out.println("3rd score: " + scores[2]);
```

```
double avg = ((double) scores[0] + scores[1]) / 2;
```

# Using Arrays

- The length variable can be accessed to get the length of the array, for example in interactions:

- ▶ `int[] scores = new int[10];`
- ▶ `scores.length`  
10

The `for` loop can be used when processing array elements

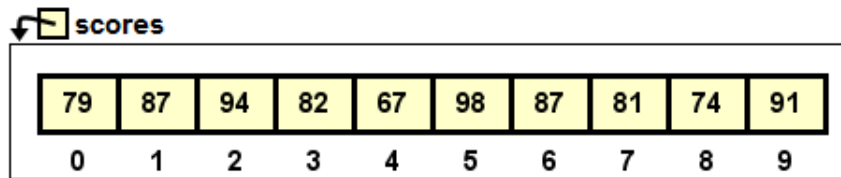
```
for (int i = 0; i < scores.length; i++) {  
    System.out.println (scores[i]);  
}
```

- The `for each` loop can also be used with arrays:

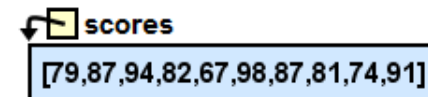
```
for (int currentScore : scores) {  
    System.out.println (currentScore);  
}
```

# Arrays

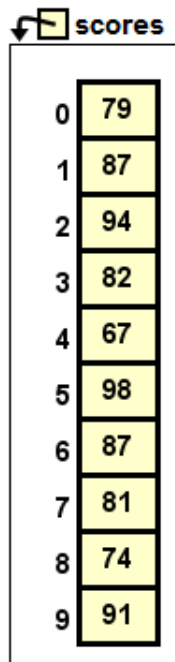
Ways to depict the `scores` array on canvas in jGRASP



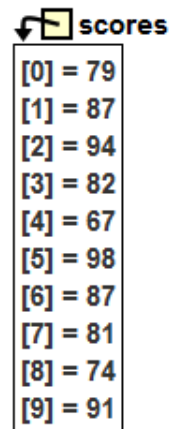
Presentation viewer



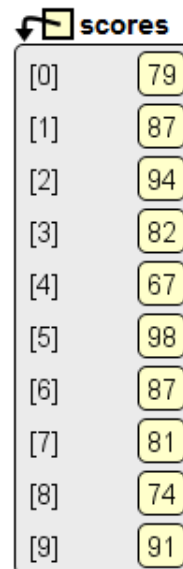
Presentation String viewer



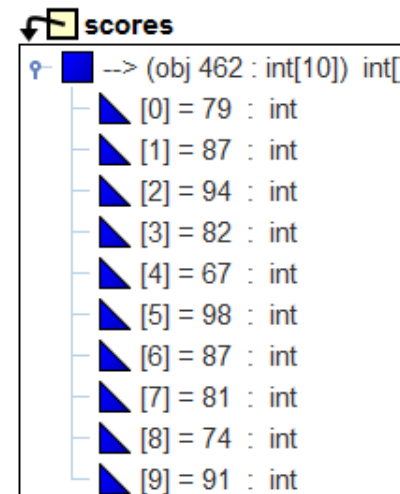
Presentation viewer (rotated)



Array Elements viewer



Basic viewer



Detail (or debug) viewer

[ScoresExample.java](#)



# Setting Array Elements

- Individual array elements are also assigned using the array name followed by the index in brackets
- Example: declare a double array and assign elements

```
double [] gradeBook = new double [4];
```

0.0	0.0	0.0	0.0
0	1	2	3

```
gradeBook[0] = 94.2;
```

94.2	0.0	0.0	0.0
0	1	2	3

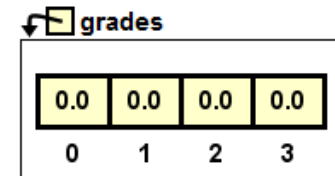
```
gradeBook[3] = 98.1;
```

94.2	0.0	0.0	98.1
0	1	2	3

# Arrays

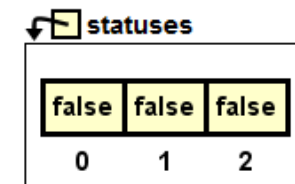
- When an array is created, the initial value of each array element depends on the type.
  - Numerical elements (including char) are initialized to zero (0, 0.0, or \0)

```
double[] grades = new double[4];
```



- boolean values are initialized to false

```
boolean[] statuses = new boolean[3];
```

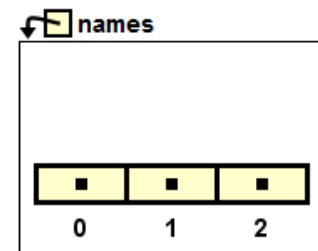


- In a reference type array, each element is initialized to null

```
String[] names = new String[3];
```

```
Coin[] change = new Coin[4];
```

```
CableAccount[] accounts = new CableAccount[3];
```



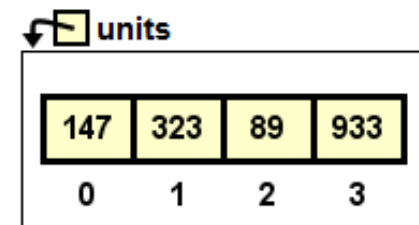
[ArrayExamples.java](#)

# Initializer Lists

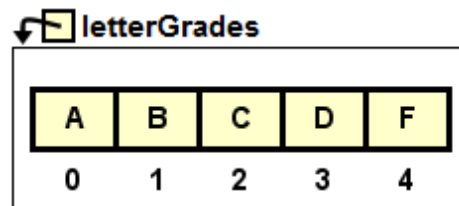
- An *initializer list* can be used to instantiate and fill an array in one step
  - The size of the array is determined by the number of items in the initializer list
  - It can only be used when declaring the array.

- Examples:

```
int[] units = {147, 323, 89, 933};
```



```
char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```



[InitializerListExample.java](#)

# Bounds Checking

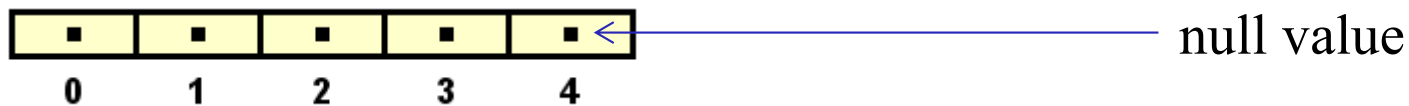
- Once an array is created, it has a fixed size
  - An index used in an array reference must specify a valid element from 0 to length - 1
- When a program runs, the Java interpreter throws an `ArrayIndexOutOfBoundsException` if an array index is out of bounds
- This is called automatic *bounds checking*
- Common in *off-by-one* errors:

```
for (int i = 0; i <= scores.length; i++) {  
    System.out.println (scores[i]);  
}
```

# More on Arrays of Objects

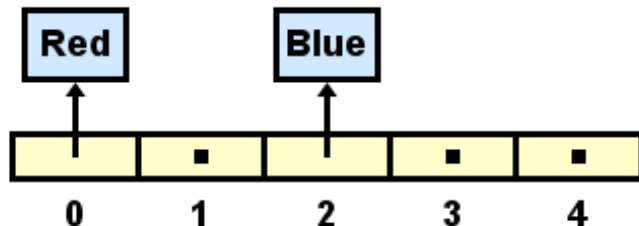
- When the elements of an array are object references, they are initialized to null (i.e., no objects are created). For example, below no String objects are created:

```
String[] colors = new String[5];
```



- Each object element stored in an array must be instantiated separately

```
colors[2] = new String("Blue");  
colors[0] = "Red"; // String objects only
```



# Arrays as Parameters

- An entire array can be passed as a parameter to a method or returned to the client program (parameters are passed by value in Java).

```
public Polygon(double[] sidesIn)
```

```
public void setSides(double[] sidesIn)
```

```
public double[] getSides()
```

- See [Polygon.java](#)
- Since parameters are passed by value, the parameter `sidesIn` becomes an alias for the array passed in

# “Aliases”

- Any reference variable passed as a parameter becomes an alias for the object passed in. This was not as important with Strings since they are immutable, but arrays and other objects can be accessed/modified via an alias so care must be exercised to avoid unexpected results.
- For example, try the following code in interactions:
  - ▶ `double[] sides1 = {5.4, 2.3, 5.7, 4.5};`
  - ▶ `Polygon shape = new Polygon(sides1);`
  - ▶ `double[] sides2 = shape.getSides();`
  - ▶ `sides2[0] = -1;`
  - ▶ `double[] sides3 = shape.getSides();`
  - ▶ `sides3[0]`  
`-1.0`

# “Aliases”

- Recall that encapsulation is achieved by objects “protecting and managing” their own information.
- If you return a reference to an array object (or any object) in a method and it is modified by a client program, does it support encapsulation?
- Lesson: be careful with reference variables as parameters and return values



# Array vs. ArrayList

- The ArrayList class has a field named `elementData` which is an array that holds the elements in the ArrayList.
- The ArrayList class provides methods for `add`, `get`, `size`, `remove`, `isEmpty`, `contains`, etc. to manage the `elementData` array
- For array types, the programmer must manage array by providing the operations above as needed
- The array is defined in most high level languages; whereas the ArrayList is provided in the Java class library, and thus is an extension to the Java language.

# Array vs. ArrayList

- Recall that the length of an array object cannot be changed. Thus, you would have to create a whole new array with the new length and copy all of the elements over.
- To insert an element at the index  $i$  of the array, you'll have to copy (move) the elements to the right to make room for the new element and increase the number of elements by one
- To delete an element at the index  $i$  in the array, you'll have to copy (move) the elements to the right of the element over one to the left and reduce the number of elements by one
- See deleteTriangle method in [TriangleList2.java](#)

# Command-Line Arguments

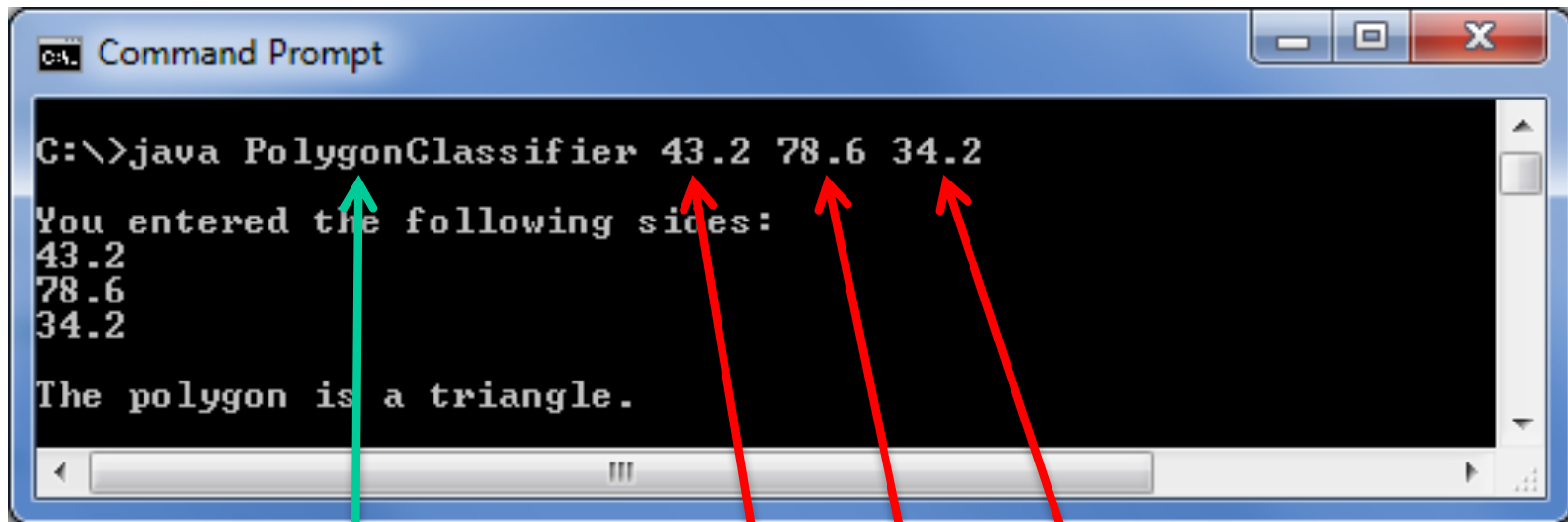
- The signature of the `main` method indicates that it takes an array of `String` objects as a parameter

```
public static void main(String[] args)
```

- The `args` array comes from *command-line arguments* that are provided when the Java interpreter is invoked (for example, in the command prompt or terminal)

# Command-Line Arguments

- Consider running the PolygonClassifier program:



```
C:\>java PolygonClassifier 43.2 78.6 34.2
You entered the following sides:
43.2
78.6
34.2
The polygon is a triangle.
```

Name of class

args[0]

args[1]

args[2]

- For ease of testing your program, command line arguments can also be passed in via jGRASP.

[PolygonClassifier.java](#)

# Variable Length Parameter Lists

- Suppose we wanted to create a method that processed a different amount of data from one invocation to the next
- For example, let's define a method called `average` that returns the average of a set of integer parameters

```
// one call to average three values
```

```
mean1 = average (42, 69, 37);
```

```
// another call to average seven values
```

```
mean2 = average (35, 43, 93, 23, 40, 21, 75);
```

# Variable Length Parameter Lists

- We could define multiple versions of the `average` method (each taking a different number of parameter inputs)
  - Downside: a separate version of the method would be needed for each parameter count
- We could define the method to accept an array of integers
  - Downside: an array would need to be created and initialized prior to calling the method each time
- Instead, Java provides a convenient way to create a *variable length parameter list*

# Variable Length Parameter Lists

- We can define a method to accept any number of parameters of the same type
- The parameters are automatically put into an array with a specified variable name

Indicates a variable length parameter list

```
public double average (int ... list)
```

↑  
element  
type

↑  
array  
name

# Variable Length Parameter Lists

```
public double average (int ... list)
{
    double result = 0.0;

    if (list.length != 0) {
        int sum = 0;
        for (int num : list) {
            sum += num;
        }
        result = (double) sum / list.length;
    }

    return result;
}
```

[VariableParams.java](#)



# Variable Length Parameter Lists

- The type of the parameter can be any primitive type or object type

```
public String allPolygons(Polygon ... polygonSet) {  
  
    String output = "";  
    for (Polygon shape : polygonSet) {  
        output += shape + " ";  
    }  
    return output;  
}
```

# Variable Length Parameter Lists

- A method that accepts a variable number of parameters can also accept other parameters
- The following method accepts an `int`, a `String` object, and a variable number of `double` values into an array called `nums`

```
public void test(int count, String name, double ... nums)
```

# Variable Length Parameter Lists

- A method can only accept one variable length parameter list
- If there are other parameters, the variable length parameter list must come last in the formal parameters
- A variable length parameter list can also be used with constructors
- See Family.java in the book.

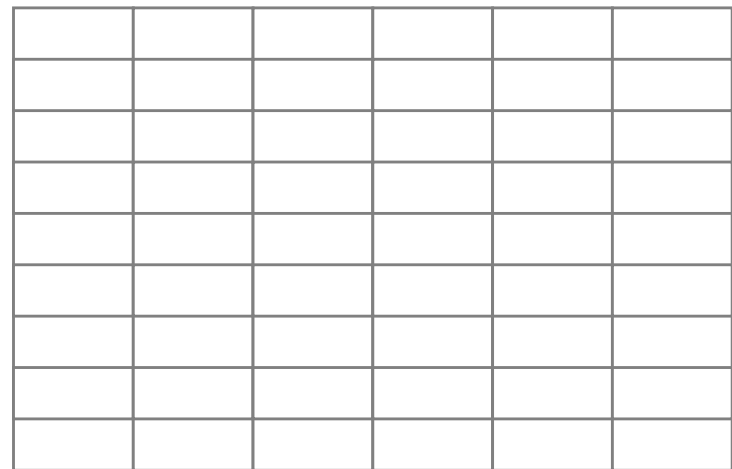
# Two-Dimensional Arrays

- A *one-dimensional array* stores a list of elements
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns

one  
dimension



two  
dimensions



# Two-Dimensional Arrays

- A two-dimensional array is an “array of arrays”
- A two-dimensional array is declared by specifying the size of each dimension separately:

```
int[][] scores = new int[12][50];
```

- A single element is referenced using two index values:

```
int value = scores[3][6];
```

- The array stored in one row can be specified using one index

```
int[] valueSet = scores[3];
```

# Two-Dimensional Arrays

Expression	Type	Description
<code>table</code>	<code>int[][]</code>	2D array of integers, or array of integer arrays
<code>table[5]</code>	<code>int[]</code>	array of integers
<code>table[5][12]</code>	<code>int</code>	integer

- Examples:

[TwoDArraySumElements.java](#)

[TwoDArraySumElementsForEach.java](#)

[TwoDArraySums.java](#)

# Multidimensional Arrays

- An array can have many dimensions – if it has more than one dimension, it is called a *multidimensional array*
- Because each dimension is an array of array references, the arrays within one dimension can be of different lengths
  - these are sometimes called *ragged arrays*

```
int [][] raggedExample = { {1, 2, 3, 4},  
                           {5, 6},  
                           {7, 8, 9} };
```