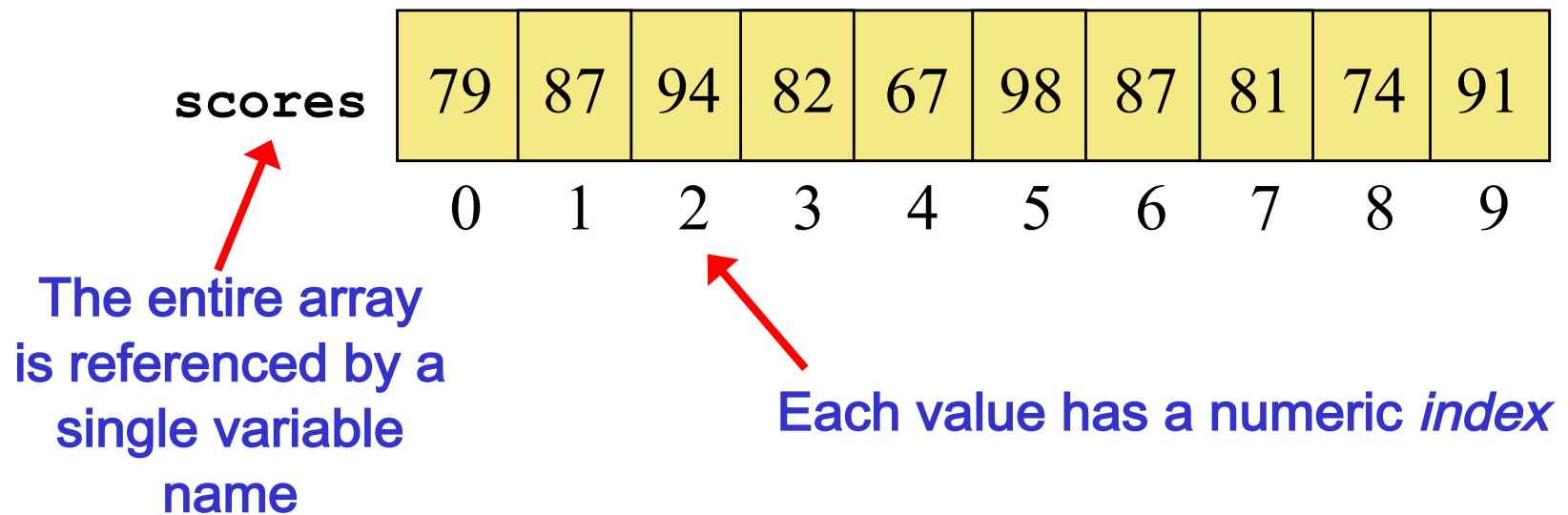# Arrays

- 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.

scores
| 79 | 87 | 94 | 82 | 67 | 98 | 87 | 81 | 74 | 91 |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

The entire array is referenced by a single variable name

Each value has a numeric *index*

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:

$$\textbf{int}[] \; \texttt{scores} = \textbf{new int}[10];$$

- The type of the variable `scores` is `int`[]
  (an array of int <u>or</u> an **int array**); when you see [], think or say array

- The reference variable `scores` is set to a new array <u>object</u> 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:

```
double[] prices;

double 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
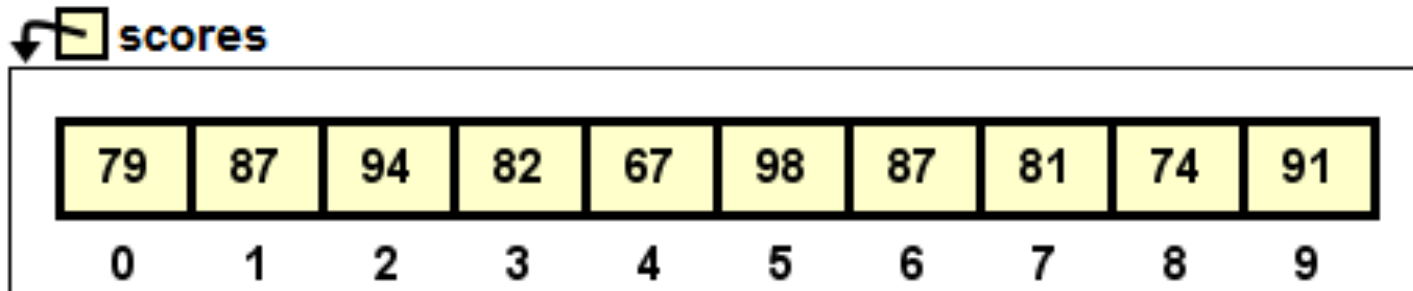
    ```
    char[] letters;
    ```

- 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

- Each array has a length identifier that 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 with the array's length 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);
  }
  ```

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;
```

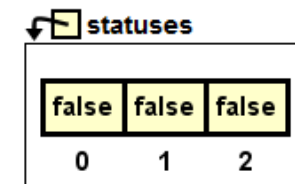| 94.2	0.0	0.0	98.1
0	1	2	3

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];
    ```

    grades

0.0	0.0	0.0	0.0
0	1	2	3
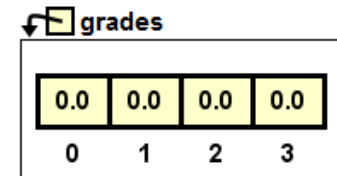
  - boolean values are initialized to false

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

    statuses

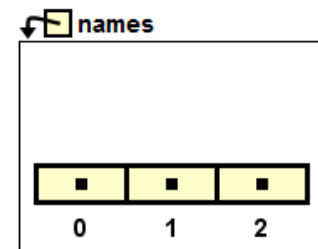 | false | false | false |
    |-------|-------|-------|
    | 0 | 1 | 2 |

  - 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];
    ```

    names

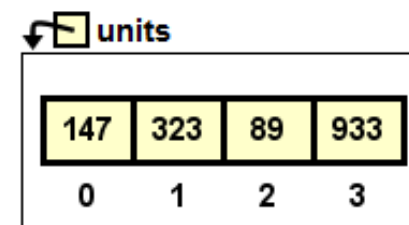 | ▪ | ▪ | ▪ |
    |---|---|---|
    | 0 | 1 | 2 |

    ArrayExamples.java
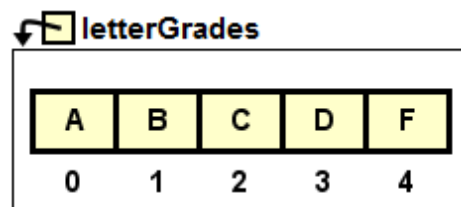
Initializer Lists

- An *initializer list* can be used to create an array and set the elements to specific values 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};
  ```
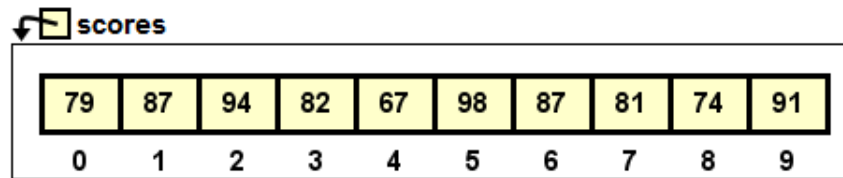


  ```
  char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
  ```
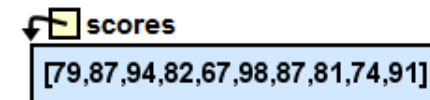

InitializerListExample.java
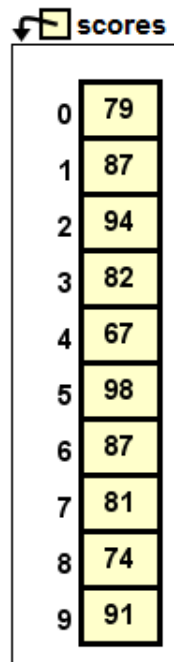
Arrays

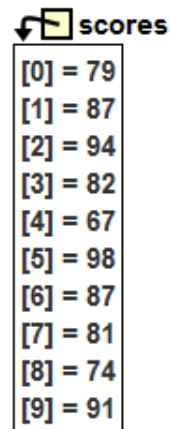## Ways to depict the `scores` array on canvas in jGRASP

scores

79	87	94	82	67	98	87	81	74	91
0	1	2	3	4	5	6	7	8	9

Presentation viewer

scores

[79,87,94,82,67,98,87,81,74,91]

Presentation String viewer

scores

0	79
1	87
2	94
3	82
4	67
5	98
6	87
7	81
8	74
9	91

Presentation viewer (rotated)

scores

[0] = 79
[1] = 87
[2] = 94
[3] = 82
[4] = 67
[5] = 98
[6] = 87
[7] = 81
[8] = 74
[9] = 91

Array Elements viewer

scores

[0] 79
[1] 87
[2] 94
[3] 82
[4] 67
[5] 98
[6] 87
[7] 81
[8] 74
[9] 91

Basic viewer

scores

--> (obj 462 : int[10]) int[]
[0] = 79 : int
[1] = 87 : int
[2] = 94 : int
[3] = 82 : int
[4] = 67 : int
[5] = 98 : int
[6] = 87 : int
[7] = 81 : int
[8] = 74 : int
[9] = 91 : int
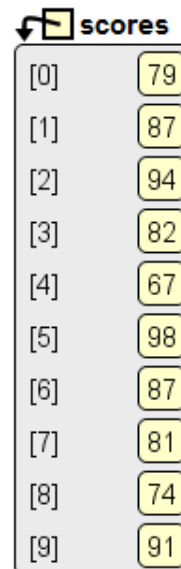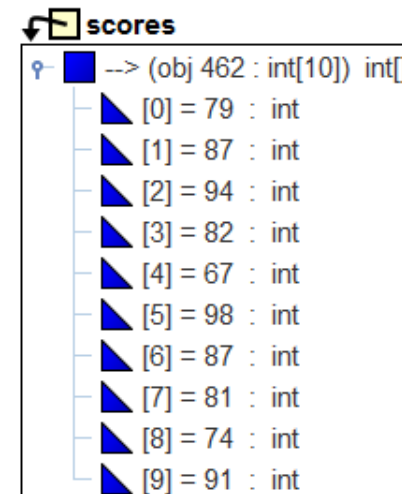
Detail (or debug) viewer

ScoresExample.java

Bounds Checking

- Once an array is created, it has a fixed length

    - An index used to reference an array element must be in the range 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:

```java
for (int i = 0; i <= scores.length; i++) {
   System.out.println (scores[i]);
}
```
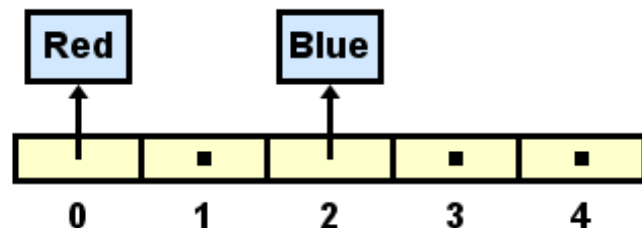
More on Arrays of Objects

- When a reference type array is created with the new operator, the array elements are initialized to null. In the example below, no String objects are created.

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


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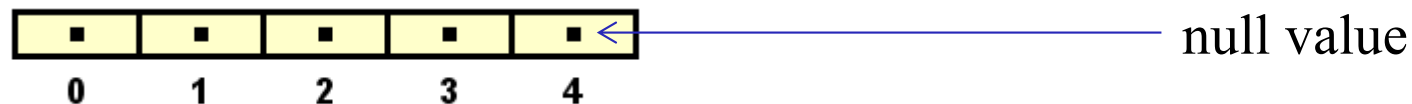
- Each object element stored in an array must be instantiated separately

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


Invoking a method on a null element will throw a **NullPointerException** e.g., `colors[1].toUpperCase()`

Arrays as Parameters

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

  ```
  public Polygon(double[] sidesIn)

  public void setSides(double[] sidesIn)

  public double[] getSides()
  ```

- See <u>Polygon.java</u>

- 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

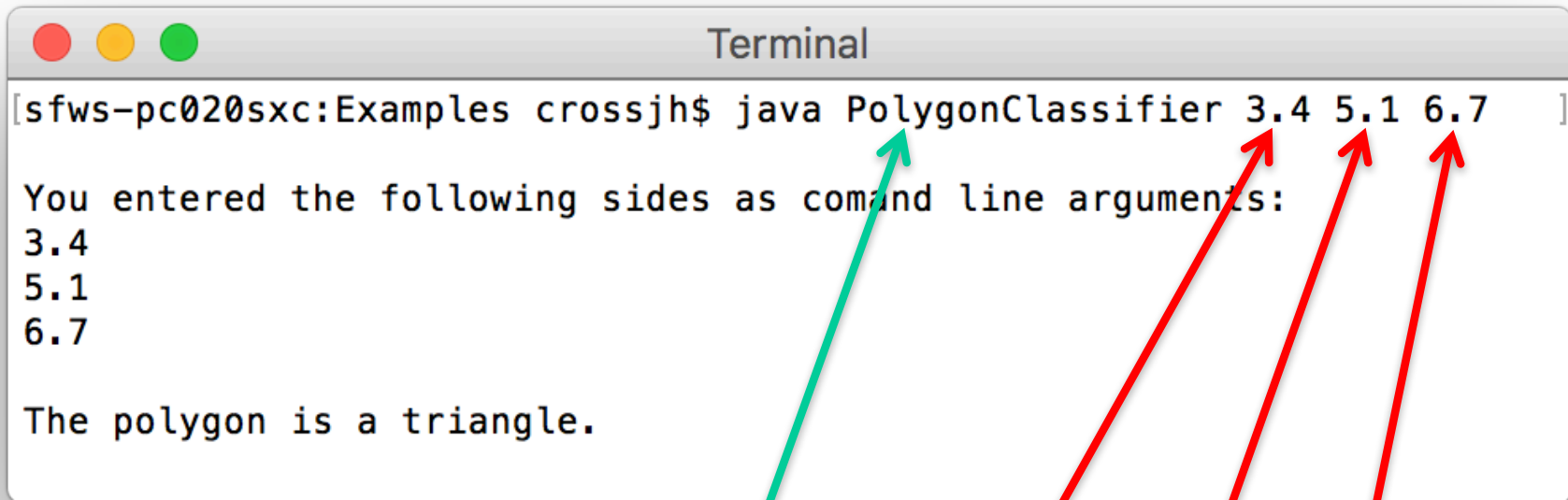- See deleteTriangle method in TriangleList2.java

Command-Line Arguments

- The `main` method 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 Java is launched (e.g., from the command line in a terminal window or from an IDE like jGRASP)

- Most professional programs allow the user to pass in arguments (data, configuration options, etc.) when the program is launched

Command-Line Arguments

- Consider running the PolygonClassifier program:

```
[sfws-pc020sxc:Examples crossjh$ java PolygonClassifier 3.4 5.1 6.7  ]

You entered the following sides as comand line arguments:
3.4
5.1
6.7

The polygon is a triangle.
```

Terminal

Name of class   **args[0]   args[1]   args[2]**

- Command-line arguments can also be passed in via jGRASP (on Build menu, turn on/off Run Arguments with check box)   PolygonClassifier.java

Variable Length Parameter Lists

- Suppose we wanted to create a method that could handle a different number of parameters 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: multiple versions of essentially the same method

- 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 that accepts 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

```java
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

```java
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 collected in an array called `nums`

```java
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

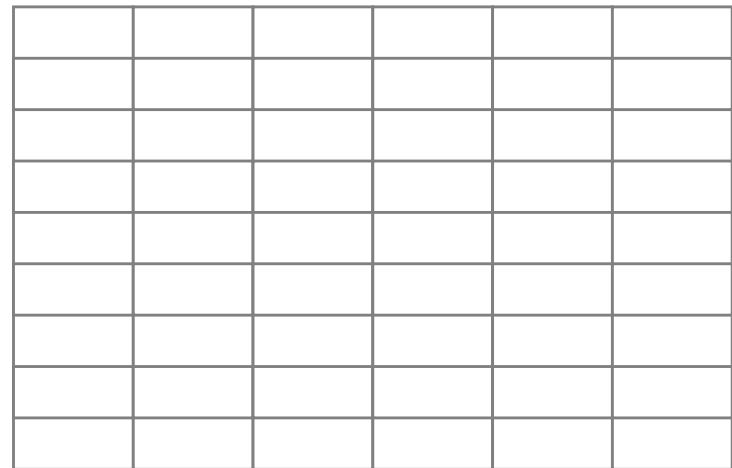- A variable length parameter list can also be used with constructors

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 (e.g., row and column):

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

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

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

Two-Dimensional Arrays

Expression	Type	Description
`table`	`int[][]`	int 2D array, 2D array of int, or array of int arrays
`table[5]`	`int[]`	int array or array of int
`table[5][12]`	`int`	int (an element in table)

- 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} };
```