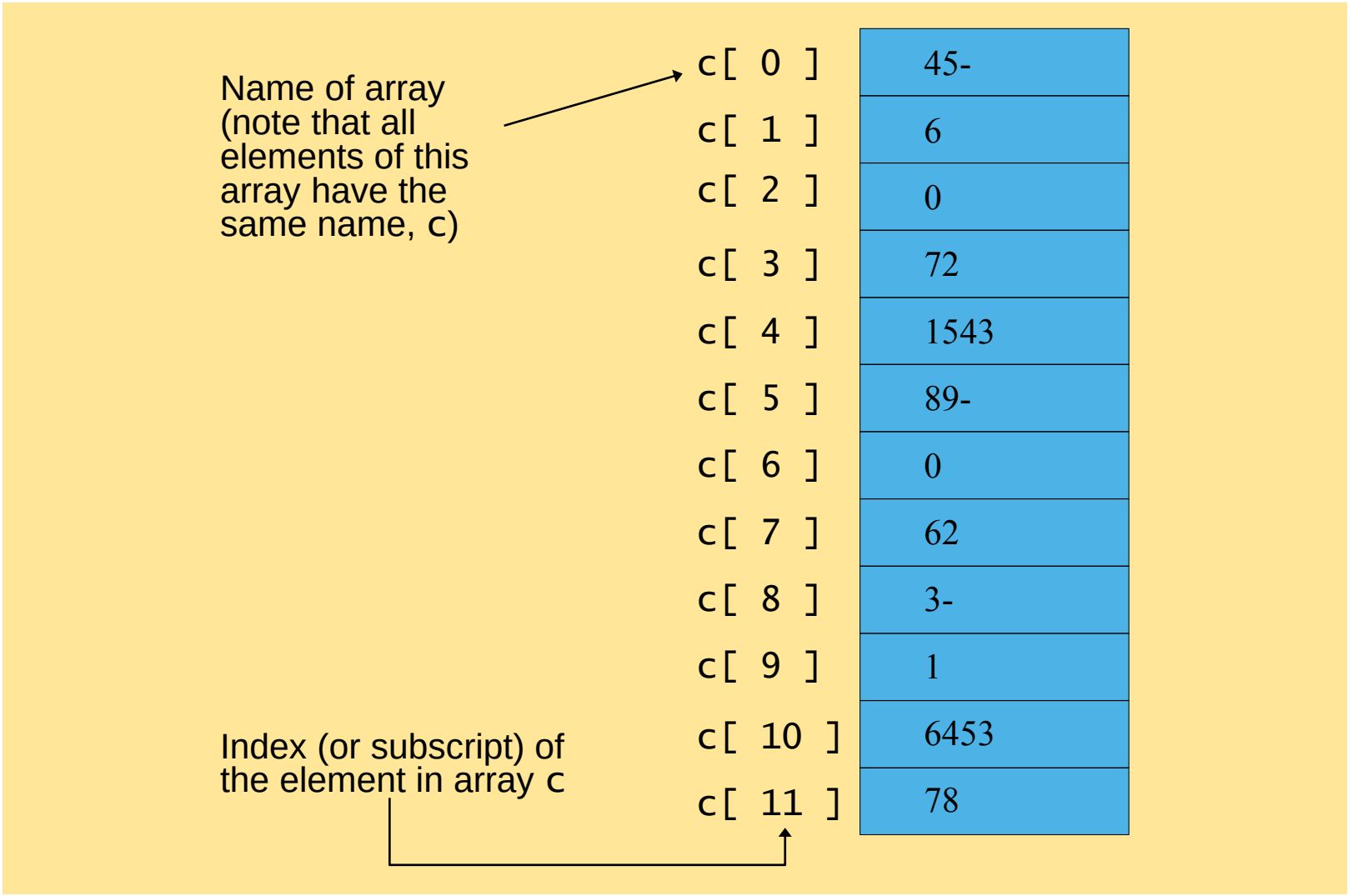
# Chapter 4 – Arrays and Strings

# Introduction

- Arrays
  - Data structures
  - Related data items of same type
  - Reference type
  - Remain same size once created
    - *Fixed-length* entries


Name of array (note that all elements of this array have the same name, c)
c[ 0 ]
c[ 1 ]
c[ 2 ]
c[ 3 ]
c[ 4 ]
c[ 5 ]
c[ 6 ]
c[ 7 ]
c[ 8 ]
c[ 9 ]
c[ 10 ]
c[ 11 ]
45-
6
0
72
1543
89-
0
62
3-
1
6453
78
Index (or subscript) of the element in array c

# Arrays (cont.)

- Index
  - Also called subscript
  - Position number in square brackets
  - Must be positive integer or integer expression

```
a = 5;
b = 6;
c[ a + b ] += 2;
```

- Adds `2` to `c[ 11 ]`

# Arrays (cont.)

- Examine array `c`
  - `c` is the array *name*
  - `c.length` accesses array `c`'s *length*
  - `c` has 12 *elements* ( `c[0]`, `c[1]`, … `c[11]` )
    - The *value* of `c[0]` is `-45`

# Declaring and Creating Arrays

- Declaring and Creating arrays
  - Arrays are objects that occupy memory
  - Created dynamically with keyword new

```
int c[] = new int[ 12 ];
```

    - Equivalent to

```
int c[];  // declare array variable
c = new int[ 12 ]; // create array
```

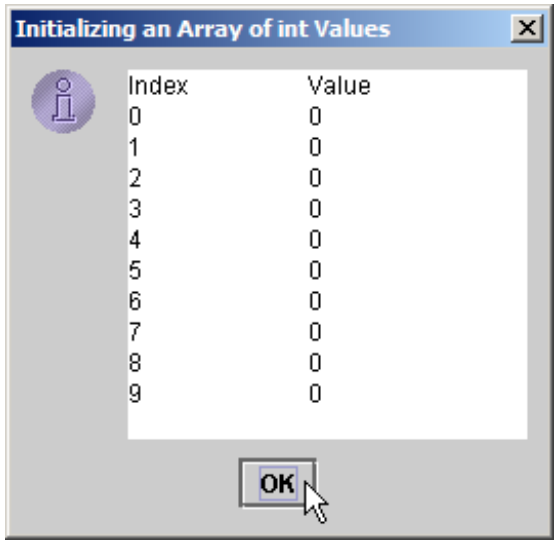
  - We can create arrays of objects too

```
String b[] = new String[ 100 ];
```

# Examples Using Arrays

- Declaring arrays
- Creating arrays
- Initializing arrays
- Manipulating array elements

```
1   // Fig. 7.2: InitArray.java
2   // Creating an array.
3   import javax.swing.*;
4
5   public class InitArray {
6
7      public static void main( String args[] )
8      {
9         int array[];            // declare reference to an array
10
11        array = new int[ 10 ];  // create array
12
13        String output = "Index\tValue\n";
14
15        // append each array element's value to String output
16        for ( int counter = 0; counter < array.length; counter++ )
17           output += counter + "\t" + array[ counter ] + "\n";
18
19        JTextArea outputArea = new JTextArea();
20        outputArea.setText( output );
21
22        JOptionPane.showMessageDialog( null, outputArea,
23           "Initializing an Array of int Values",
24           JOptionPane.INFORMATION_MESSAGE );
25
26        System.exit( 0 );
27
28     } // end main
29
30  } // end class InitArray
```



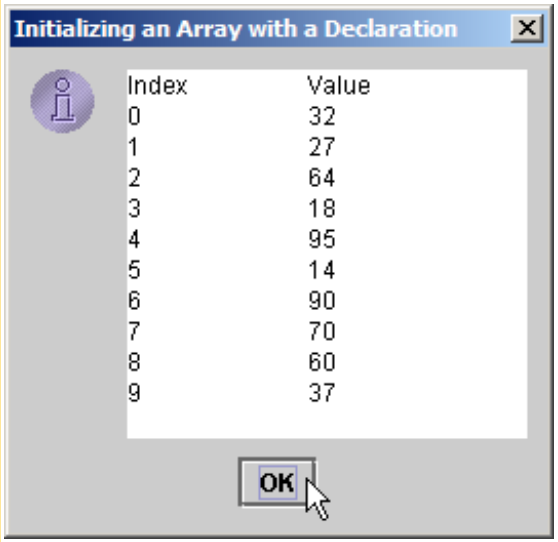
# Examples Using Arrays (Cont.)

- Using an array initializer
  - Use *initializer list*
    - Items enclosed in braces (`{}`)
    - Items in list separated by commas

      ```
      int n[] = { 10, 20, 30, 40, 50 };
      ```

      - Creates a five-element array
      - Index values of 0, 1, 2, 3, 4
  - Do not need keyword `new`

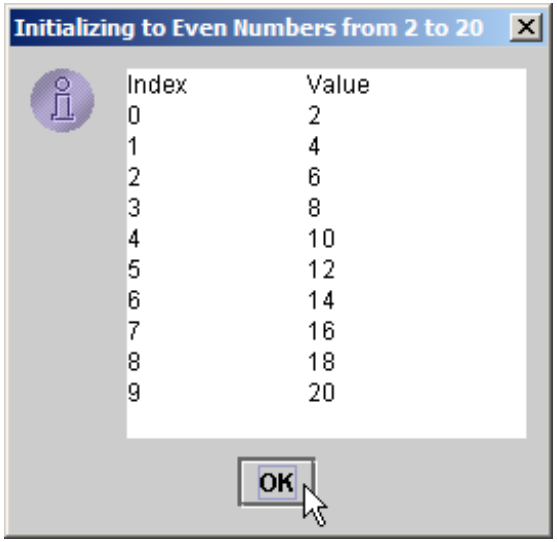
```
1      // Fig. 7.3: InitArray.java
2      // Initializing an array with a declaration.
3      import javax.swing.*;
4
5      public class InitArray {
6
7         public static void main( String args[] )
8         {
9            // array initializer specifies number of elements and
10           // value for each element
11           int array[] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };
12
13           String output = "Index\tValue\n";
14
15           // append each array element's value to String output
16           for ( int counter = 0; counter < array.length; counter++ )
17              output += counter + "\t" + array[ counter ] + "\n";
18
19           JTextArea outputArea = new JTextArea();
20           outputArea.setText( output );
21
22           JOptionPane.showMessageDialog( null, outputArea,
23              "Initializing an Array with a Declaration",
24              JOptionPane.INFORMATION_MESSAGE );
25
26           System.exit( 0 );
27
28        } // end main
29     } // end class InitArray
```



# Examples Using Arrays (Cont.)

- Calculating the value to store in each array element
  - Initialize elements of 10-element array to even integers

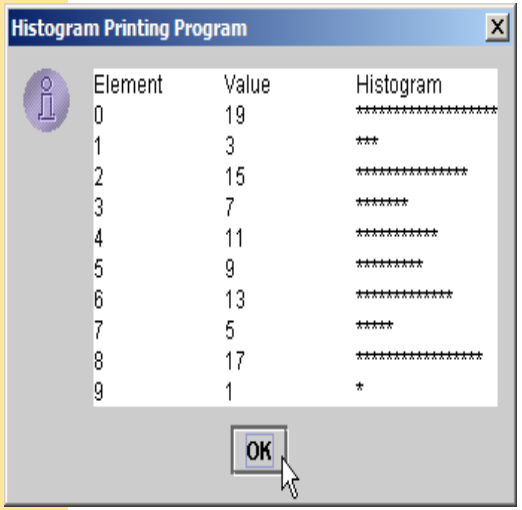
```
// Initialize array with the even integers from 2 to 20.
  import javax.swing.*;
    public class InitArray {
       public static void main( String args[]) {
        final int ARRAY_LENGTH = 10;     // constant
       int array[];                      // reference to int array
        array = new int[ ARRAY_LENGTH ];  // create array
        for ( int counter = 0; counter < array.length; counter++ )
                array[ counter ] = 2 + 2 * counter;
           String output = "Index\tValue\n";
        for ( int counter = 0; counter < array.length; counter++ )
          output += counter + "\t" + array[ counter ] + "\n";
       JTextArea outputArea = new JTextArea();
        outputArea.setText( output );
       JOptionPane.showMessageDialog( null, outputArea,
          "Initializing to Even Numbers from 2 to 20",
          JOptionPane.INFORMATION_MESSAGE );

       System.exit( 0 );
    } // end main
} // end class InitArray
```



# Examples Using Arrays (Cont.)

- Using histograms do display array data graphically
  - Histogram
    - Plot each numeric value as bar of asterisks (*)

```
// Histogram printing program.
import javax.swing.*;
  public class Histogram {
     public static void main( String args[] ) {
      int array[] = { 19, 3, 15, 7, 11, 9, 13, 5, 17, 1 };

     String output = "Element\tValue\tHistogram";
      // for each array element, output a bar in histogram
     for ( int counter = 0; counter < array.length; counter++ ) {
        output += "\n" + counter + "\t" + array[ counter ] + "\t";
          for ( int stars = 0; stars < array[ counter ]; stars++ )
           output += "*";
      } // end outer for

    JTextArea outputArea = new JTextArea();
    outputArea.setText( output );
  JOptionPane.showMessageDialog( null, outputArea,
            "Histogram Printing Program",
            JOptionPane.INFORMATION_MESSAGE );

    System.exit( 0 );

  } // end main

} // end class Histogram
```



# Examples Using Arrays (Cont.)

- Some additional points
  - When looping through an array
    - Index should never go below 0
    - Index should be less than total number of array elements
  - When invalid array reference occurs
    - Java generates `ArrayIndexOutOfBoundsException`

# References and Reference Parameters

- Two ways to pass arguments to methods
  - Pass-by-value
    - Copy of argument's value is passed to called method
    - In Java, every primitive is pass-by-value
  - Pass-by-reference
    - Caller gives called method direct access to caller's data
    - Called method can manipulate this data
    - Improved performance over pass-by-value
    - In Java, every object is pass-by-reference
      - In Java, arrays are objects
        - Therefore, arrays are passed to methods by reference

# Passing Arrays to Methods

- To pass array argument to a method
  - Specify array name without brackets
    - Array `hourlyTemperatures` is declared as

```
int hourlyTemperatures = new int[ 24 ];
```

    - The method call

```
modifyArray( hourlyTemperatures );
```

    - Passes array `hourlyTemperatures` to method `modifyArray`

```java
// Example Passing array
    public class PassArray
    {

       public static void main( String args[] )
       {
          int array[] = { 1, 2, 3, 4, 5 };

          System.out.println(
             "Effects of passing reference to entire array:\n" +
             "The values of the original array are:" );


          for ( int value : array )
             System.out.printf( "   %d", value );

          modifyArray( array ); // pass array reference
          System.out.println( "\n\nThe values of the modified array are:" );

          // output modified array elements
          for ( int value : array )
             System.out.printf( "   %d", value );

          System.out.printf(
             "\n\nEffects of passing array element value:\n" +
             "array[3] before modifyElement: %d\n", array[ 3 ] );
```

```
      modifyElement( array[ 3 ] ); // attempt to modify array[ 3 ]
         System.out.printf(
            "array[3] after modifyElement: %d\n", array[ 3 ] );
   } // end main


      public static void modifyArray( int array2[] )
      {
         for ( int counter = 0; counter < array2.length; counter++ )
            array2[ counter ] *= 2;
      } // end method modifyArray

      // multiply argument by 2
      public static void modifyElement( int element )
      {
         element *= 2;
         System.out.printf(
            "Value of element in modifyElement: %d\n", element );
      } // end method modifyElement
   } // end class PassArray
```

```
Effects of passing reference to entire array:
The values of the original array are:
   1   2   3   4   5

The values of the modified array are:
   2   4   6   8   10

Effects of passing array element value:
array[3] before modifyElement: 8
Value of element in modifyElement: 16
array[3] after modifyElement: 8
```

# Multidimensional Arrays

- Multidimensional arrays
  - Tables with rows and columns
    - Two-dimensional array
    - Declaring two-dimensional array `b[2][2]`

      ```
      int b[][] = { { 1, 2 }, { 3, 4 } };
      ```

      - `1` and `2` initialize `b[0][0]` and `b[0][1]`
      - `3` and `4` initialize `b[1][0]` and `b[1][1]`

      ```
      int b[][] = { { 1, 2 }, { 3, 4, 5 } };
      ```

      - row `0` contains elements `1` and `2`
      - row `1` contains elements `3`, `4` and `5`

# Multidimensional Arrays (Cont.)

- Creating multidimensional arrays
  - Can be allocated dynamically
    - 3-by-4 array

```
int b[][];
b = new int[ 3 ][ 4 ];
```

    - Rows can have different number of columns

```
int b[][];
b = new int[ 2 ][ ];   // allocate rows
b[ 0 ] = new int[ 5 ]; // allocate row 0
b[ 1 ] = new int[ 3 ]; // allocate row 1
```

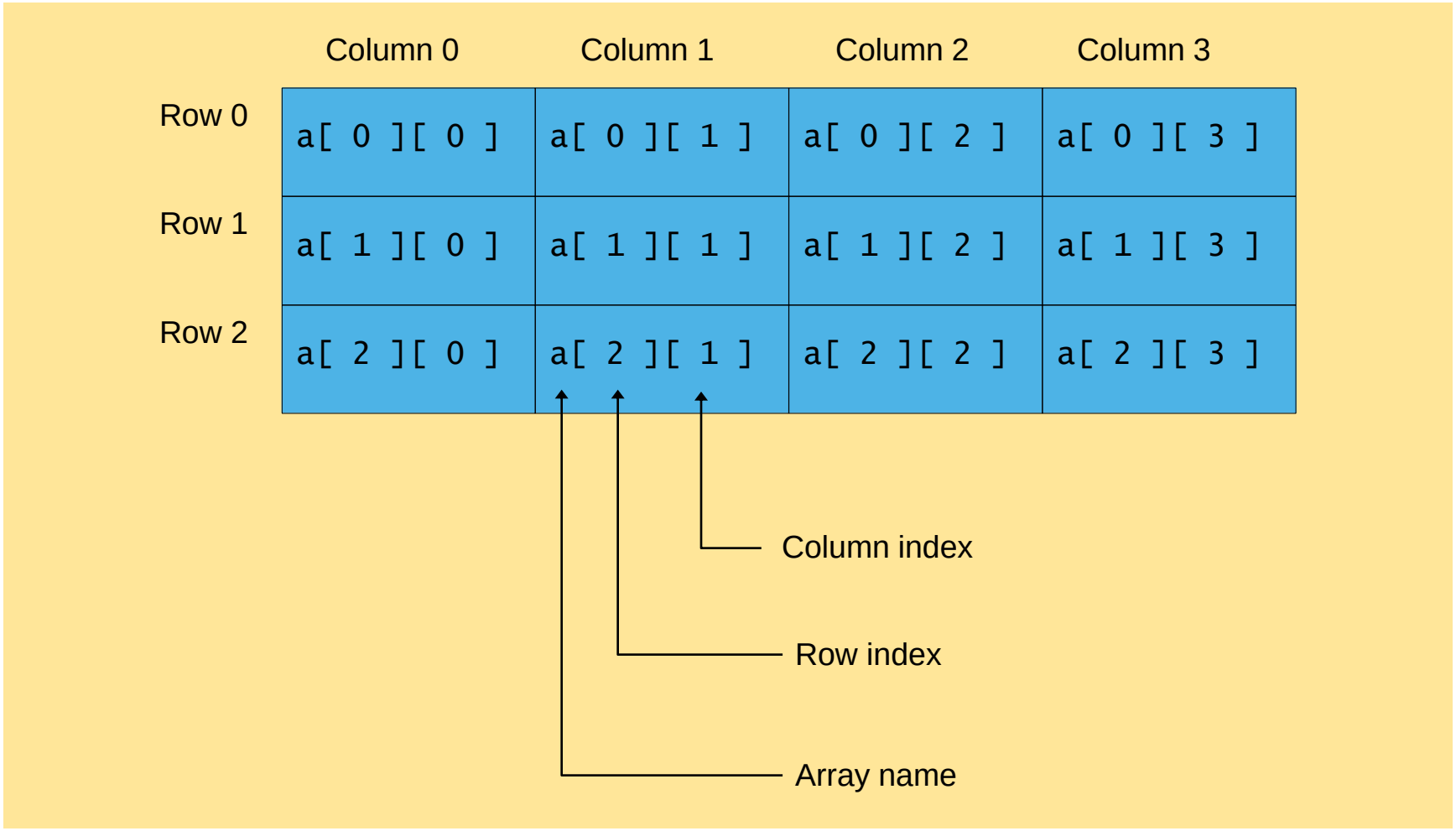
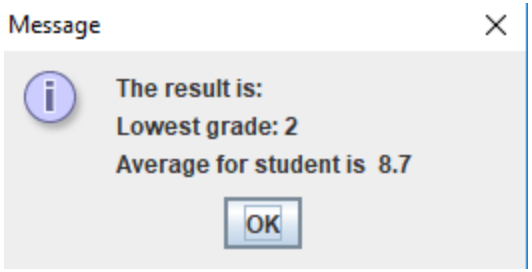



**Fig. 7.13 Two-dimensional array with three rows and four columns.**




```
import javax.swing.JOptionPane;
public class TryOne {
    public static void main(String[] args) {

        int grades[][] = { { 10, 10, 20 },
                           { 5, 5, 5, 10 },
                           { 10, 2, 10} };
       String output;
       output = "The result is:\n";
       output += "Lowest grade: " + minimum(grades);
       output += "\nAverage for student is  " + average( grades );
       JOptionPane.showMessageDialog(null,output);  }

 public static int minimum(int xgrades[][]) {
      int lowGrade = xgrades[ 0 ][ 0 ];
        for ( int row = 0; row < xgrades.length; row++ )
            for ( int column = 0; column < xgrades[row].length; column++ )
                if ( xgrades[ row ][ column ] < lowGrade )
                           lowGrade = xgrades[ row ][ column ];
                return lowGrade; }

public static double average( int ygrades[][] ) {
               int total = 0;
               int count = 0;
        for ( int row = 0; row < ygrades.length; row++ )
           for ( int column = 0; column < ygrades[row].length; column++ ){
                        total += ygrades[row][column];
                        ++count; }
               return ( double ) total / count;  } }
```

# Strings

- Contents
  - Strings are objects, immutable, differ from arrays
  - Basic methods on Strings
  - Convert String representation of numbers into numbers
  - StringBuffer, mutable version of Strings

# String

- **Java.lang.String**
- **Java.lang.StringBuffer**
- `String` is an **object**
- Creating a `String`
  - form string literal between double quotes

```
String s = "Hello,
  World!";
```

  - by using the `new` keyword

```
String s = new
  String("Java");
```

# Strings

- **`String`** is a class (**`java.lang.String`**) offering methods for almost anything

```
String s = "a string literal";
```

- + is concatenation, when you concatenate a string with a value that is not a string, the latter is converted to a string

```
s = "The year is " + 2002 + "!";
```

- Everything can be converted to a string representation including primitive types and objects

# Strings

- **string**: An object storing a sequence of text characters.
  - Unlike most other objects, a `String` is not created with `new`.

```
String name = "text";
```

  - Examples:

```
String name = "Mohammad Ali";
int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

# Indexes

- Characters of a string are numbered with 0-based *indexes*:

```
String name = "P. Diddy";
```

| index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| char | P | . |  | D | i | d | d | y |

  - The first character's index is always 0
  - The last character's index is 1 less than the string's length
  - The individual characters are values of type `char`

# `String` methods

| Method name | Description |
|---|---|
| `indexOf(`**`str`**`)` | index where the start of the given string appears in this string (-1 if it is not there) |
| `length()` | number of characters in this string |
| `substring(`**`index1`**`, `**`index2`**`)`<br>or<br>`substring(`**`index1`**`)` | the characters in this string from *index1* (inclusive) to *index2* (exclusive);<br>if *index2* omitted, grabs till end of string |
| `toLowerCase()` | a new string with all lowercase letters |
| `toUpperCase()` | a new string with all uppercase letters |

- These methods are called using the dot notation:

```
String sta = "Dr. Dre";
System.out.println(sta.length());    // 7
```

# `String` method examples

```
String s1 = "Stuart Reges";
String s2 = "Marty Stepp";
System.out.println(s1.length());          // 12
System.out.println(s1.indexOf("e"));      // 8
System.out.println(s1.substring(7, 10))   //Reg

String s3 = s2.substring(2, 8);
System.out.println(s3.toLowerCase()); //rty st
```

# Modifying strings

- Methods like `substring`, `toLowerCase`, etc. create/return
  a new string, rather than modifying the current string.

```
String s = "lil bow wow";
s.toUpperCase();
System.out.println(s);   // lil bow wow
```

- To modify a variable, you must reassign it:

```
String s = "lil bow wow";
s = s.toUpperCase();
System.out.println(s);   // LIL BOW WOW
```

# Strings as parameters

```
public class StringParameters {
    public static void main(String[] args) {
        sayHello("Marty");

        String teacher = "Helene";
        sayHello(teacher);
    }

    public static void sayHello(String name) {
        System.out.println("Welcome, " + name);
    }
}
```

Output:

```
Welcome, Marty
Welcome, Helene
```

# Strings as user input

- **`Scanner`'s `next`** method reads a word of input as a **`String`**.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
name = name.toUpperCase();
System.out.println(name + " has " + name.length() +
" letters and starts with " + name.substring(0,
1));
```

Output:

```
What is your name? Madonna
MADONNA has 7 letters and starts with M
```

- The **`nextLine`** method reads a line of input as a **`String`**.

```
System.out.print("What is your address? ");
String address = console.nextLine();
```

# Comparing strings

- Relational operators such as < and == fail on objects.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name == "Barney") {
    System.out.println(“hello, how are you?,");
    System.out.println("We're a happy family!");
}
```

- This code will compile, but it will not print the song.

- == compares objects by *references* , so it often gives **`false`** even when two **`Strings`** have the same letters.

# The `equals` method

- Objects are compared using a method named **`equals`**.

```
Scanner console = new Scanner(System.in);
System.out.print("What is your name? ");
String name = console.next();
if (name.equals("Barney")) {
    System.out.println(“hello, how are you?,");
    System.out.println("We're a happy family!");
}
```

  - Technically this is a method that returns a value of type `boolean`, the type used in logical tests.

# `String` test methods

| Method | Description |
|---|---|
| `equals(`**str**`)` | whether two strings contain the same characters |
| `equalsIgnoreCase(`**str**`)` | whether two strings contain the same characters, ignoring upper vs. lower case |
| `startsWith(`**str**`)` | whether one contains other's characters at start |
| `endsWith(`**str**`)` | whether one contains other's characters at end |
| `contains(`**str**`)` | whether the given string is found within this one |

```
String name = console.next();
if (name.startsWith("Dr.")) {
    System.out.println("Are you single?");
} else if (name.equalsIgnoreCase("MANGER")) {
    System.out.println("I need your name.");
}
```

# Type `char`

- **`char`** : A primitive type representing single characters.
  - Each character inside a `String` is stored as a `char` value.
  - Literal `char` values are surrounded with apostrophe (single-quote) marks, such as `'a'` or `'4'` or `'\n'` or `'\''`
  - It is legal to have variables, parameters, returns of type `char`

```
char letter = 'S';
System.out.println(letter);  // S
```

- `char` values can be concatenated with strings.

```
char initial = 'P';
System.out.println(initial + " Diddy");
// P Diddy
```

# The `charAt` method

- The `chars` in a `String` can be accessed using the `charAt` method.

```
String food = "cookie";
char firstLetter = food.charAt(0);    // 'c'

System.out.println(firstLetter + " is for " + food);
System.out.println("That's good enough for me!");
```

- You can use a `for` loop to print or examine each character.

```
String major = "CSE";
for (int i = 0; i < major.length(); i++) {
    char c = major.charAt(i);
    System.out.println(c);
}
```

Output:

```
C
S
E
```

# `char` vs. `int`

- All **`char`** values are assigned numbers internally by the computer, called *ASCII* values.
  - Examples:
    `'A'` is 65, `'B'` is 66, `' '` is 32
    `'a'` is 97, `'b'` is 98, `'*'` is 42
  - Mixing `char` and `int` causes automatic conversion to `int`.
    `'a' + 10` is 107, `'A' + 'A'` is 130
  - To convert an `int` into the equivalent `char`, type-cast it.
    `(char) ('a' + 2)` is `'c'`

# char vs. String

- "h" is a String
  'h' is a char (the two behave differently)

- String is an object; it contains methods

```
String s = "h";
s = s.toUpperCase();          // 'H'
int len = s.length();         //  1
char first = s.charAt(0);     // 'H'
```

- char is primitive; you can't call methods on it

```
char c = 'h';
c = c.toUpperCase();    // ERROR: "cannot be
  dereferenced"
```

# Comparing `char` values

- You can compare `char` values with relational operators:

```
'a' < 'b'  and  'X' == 'X'  and  'Q' != 'q'
```

  - An example that prints the alphabet:

```
for (char c = 'a'; c <= 'z'; c++) {
    System.out.print(c); }
```

- You can test the value of a string's character:

```
String word = console.next();
if (word.charAt(word.length() - 1) == 's') {
    System.out.println(word + " is plural.");
   }
```

# `StringBuffer`

- The **`String`** class is used for constant strings
- While **`StringBuffer`** is for strings that can change
- **`StringBuffer`** contains a method **`tostring`**( ) which returns the string value being held
- Since **`String`** is *immutable*, it is "*cheaper*"!

# StringBuffer

```
public class StringEx{
public static void main(String argv[]) {

StringBuffer sb = new StringBuffer("Drink Java!");
StringBuffer prev_sb = sb;
sb.insert(6, "Hot ");
sb.append(" Cheer!");

System.out.println(sb.toString() + " : " + sb.capacity());
System.out.println("prev_sb becomes " + prev_sb );} }
```

```
****  output *****
Drink Hot Java! Cheer! : 27 (initial size + 16)
prev_sb becomes Drink Hot Java! Cheer!
```