CSEN 2302 : Programming II

Review of CSEN 2302 / CSE 102

Lecture Objectives

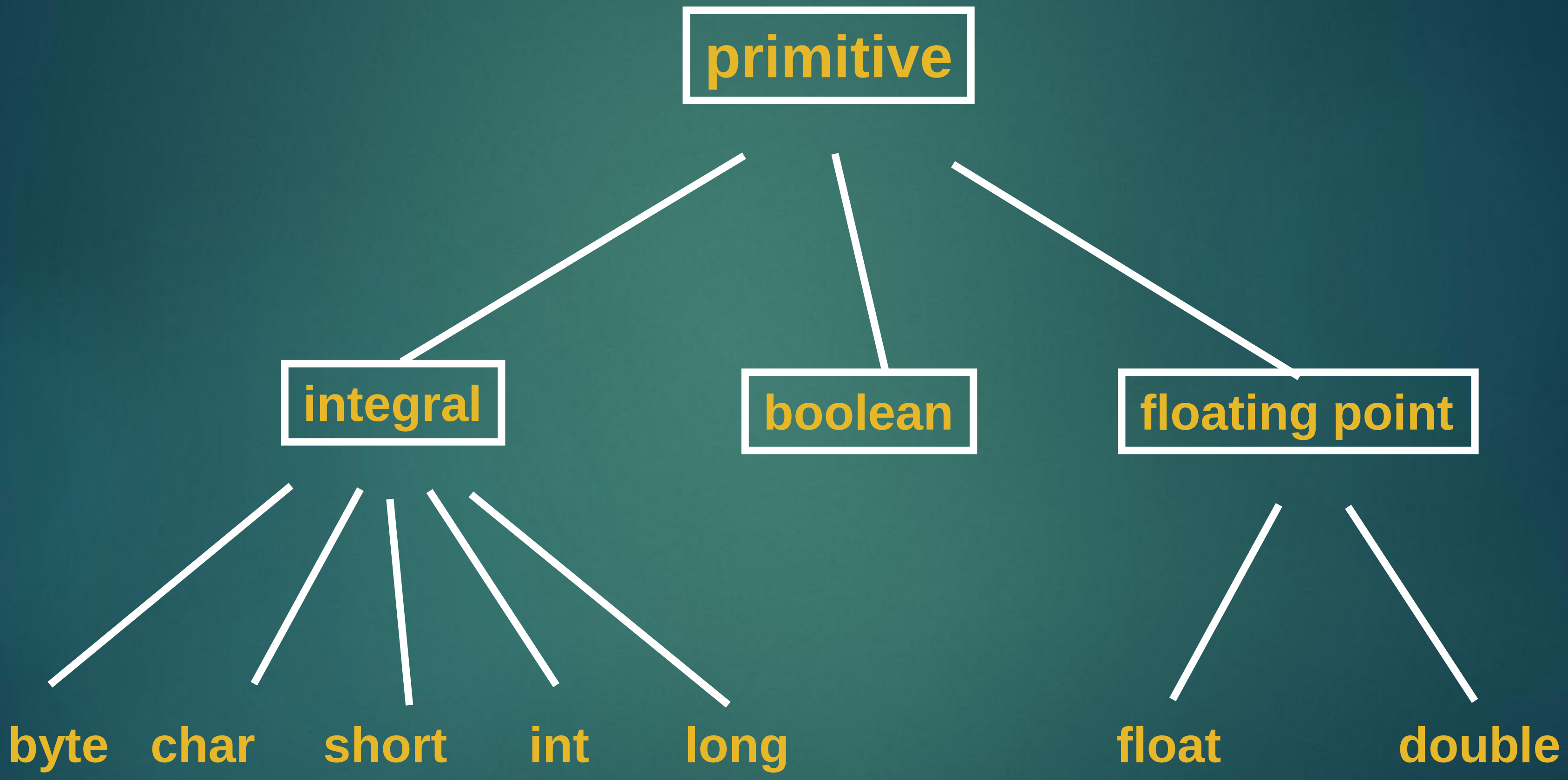
- To review the major topics covered in **CSEN 2302 / CSE 102** course
- Refresh the memory and get ready for the course **CSEN 2302 / CSE 102**

Outline

Quick Review of CSEN 2302 / CSE 102

- Primitive and Reference Types
- Initializing Class Variables
- Defining Constructors
- How to Create a String
- How to Perform Operations on Strings
- Arrays

Java Primitive **Data Types**



Cont…

Type	Size in Bits	Minimum Value	to	Maximum Value
byte	8	-128	to	127
short	16	-32,768	to	32,767
int	32	-2,147,483,648	to	2,147,483,647
long	64	-9,223,372,036,854,775,808	to	+9,223,372,036,854,775,807
float	32	$\pm$1.4E - 45	to	$\pm$3.4028235E+38
double	64	$\pm$4.9E - 324	to	+1.7976931348623157E+308

Simple **Initialization** of Instance Variables

- Instance variables can be initialized **at declaration.**

```
String name = “CSE 201”;
```

- Initialization happens **at object creation.**

```
public class Movie {
  private String title;
  private String rating = "G";
  private int numOfOscars = 0;
…
}
```

- More complex initialization should be placed in a **constructor**.

Defining Constructors

```
public class Movie {
  private String title;
  private String rating = "PG";

  public Movie() {
    title = "Last Action …";
  }
  public Movie(String newTitle) {
    title = newTitle;
  }
}
```

The Movie class now provides two constructors.

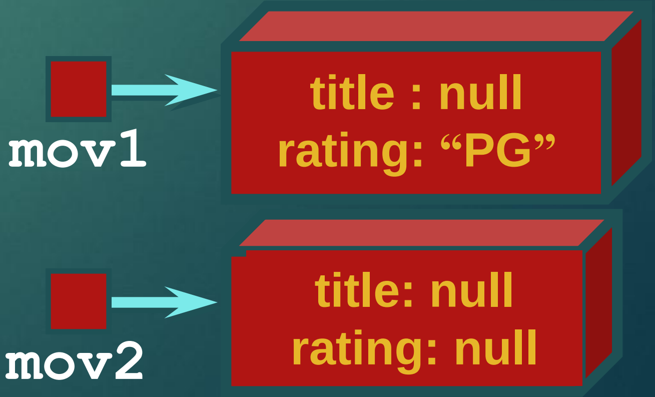
```
Movie mov1 = new Movie();
Movie mov2 = new Movie("Gone …");
Movie mov3 = new Movie("The Good …");
```

The **this** Reference

- Instance methods receive an argument called `this`, which refers to the current object.

```
public class Movie {
  private String title;
  private String rating;
  public void setRating(String newRating) {
    this.rating = newRating;
  }
  …
}
```

```
void anyMethod() {
 Movie mov1 = new Movie();
 mov1.setRating("PG");
 Movie mov2 = new Movie();
 …
```



Sharing Code Between Constructors

```
public class Movie {
  private String title;
  private String rating;

  public Movie() {
    this("G");
  }
  public Movie(String newRating) {
    rating = newRating;
  }
}
```

A constructor can call another constructor by using `this()`.

```
Movie mov2 = new Movie();
```

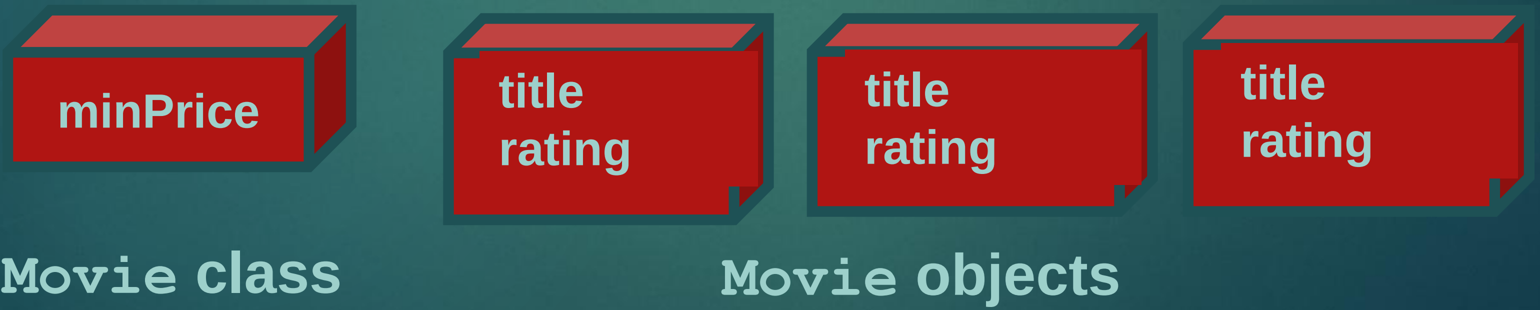
What happens here?

Class Variables

- Class variables belong to a class and are common to all instances of that class.

- Class variables are declared as static in class definitions.

```
public class Movie {
  private static double minPrice;    // class var
  private String title, rating;      // inst vars
```



Initializing Class Variables

- Class variables can be initialized at declaration.
- Initialization takes place when the class is loaded.

```
public class Movie {
  private static double minPrice = 1.29;

  private String title, rating;
  private int length = 0;
```

Class Methods

- Class methods are shared by all instances.
- Useful for manipulating class variables:

```
public static void increaseMinPrice(double inc) {
  minPrice += inc;
}
```

- Call a class method by using the class name or an object reference.

```
Movie.increaseMinPrice(0.50);
mov1.increaseMinPrice(0.50);
```

Garbage Collection

- Memory management in Java is automatic.
- When all references to an object are lost, it is marked for garbage collection.
  - **Garbage collection reclaims memory used by the object.**
- Garbage collection is automatic.
  - There is no need for the programmer to do anything.

How to **Create a String**

- Assign a double-quoted constant to a `String` variable:

```
String category = "Action";
```

- Concatenate other strings:

```
String empName = firstName + " " + lastName;
```

- Use a constructor:

```
String empName = new String("Joe Smith");
```

How to **Concatenate Strings**

- Use the + operator to concatenate strings:

```
System.out.println("Name = " + empName);
```

- You can concatenate primitives and strings:

```
int age = getAge();
System.out.println("Age = " + age);
```

- **`String.concat()`** is another way to concatenate strings.

How to Perform **Operations on Strings**

- How to find the **length** of a string:

```
int length();
```

```
String str = "Comedy";
int len = str.length();
```

- How to find the **character** at a specific index:

```
char charAt(int index);
```

```
String str = "Comedy";
char c = str.charAt(1);
```

- How to return a **substring** of a string:

```
String substring(int beginIndex,int endIndex);
```

```
String str = "Comedy";
String sub = str.substring(2,4);
```

What will be displayed by me? ***System.out.println(sub);***

How to Perform Operations on Strings (Cont.)

- How to convert to **uppercase or lowercase:**

```
String toUpperCase();
String toLowerCase();
```

```
String caps =
      str.toUpperCase();
```

- How to **Trim** whitespace:

```
String trim();
```

```
String nospaces = str.trim();
```

- How to find the **index** of a substring:

```
int indexOf (String str);
int lastIndexOf(String str);
```

```
String str = "Comedy";
int index = str.indexOf("me");
```

How to **Compare Two Strings**

- Use **`equals()`** if you want font case to count:

```
String passwd = connection.getPassword();
if ( passwd.equals("fgHPUw") )… // Case is important
```

- Use **`equalsIgnoreCase()`** if you want to ignore font case:

```
String cat = getCategory();
if (cat.equalsIgnoreCase("Drama"))…
                    // We just want the word to match
```

- **Do not use `==`.**

Cont...

Method Name	Parameter	Returns Type	Operation Performed
equals	String	boolean	Tests for equality of string contents.
compareTo	String	int	Returns 0 if equal, a positive integer if the string in the parameter comes before the string associated with the method and a negative integer if the parameter comes after it.

What will be displayed by:

```
String s = "A";
System.out.println(s.compareTo("B") );
```

How to Produce **Strings from Other Objects**

- Use **`Object.toString()`.**
- Your class can **override** the method `toString()`:

```
public Class Movie {…
      public String toString() {
            return name + " (" + Year + ")";
      }…
```

- **`System.out.println()`** automatically calls an object's `toString()` method:

```
Movie mov = new Movie(…);
System.out.println("Title Rented: " + mov);
```

What value is returned?

```
// Using methods length, indexOf, substring


    String  stateName = “Mississippi” ;

    stateName.length( )

    stateName.indexOf(“is”)

    stateName.substring( 0, 4 )

    stateName.substring( 4, 6 )

    stateName.substring( 9, 11 )
```

What value is returned? (Cont’d)

```
// Using methods length, indexOf, substring


      String  stateName = “Mississippi” ;

      stateName.length( )                         value 11
      stateName.indexOf(“is”)                     value 1
      stateName.substring( 0, 4 )                 value “Miss”
      stateName.substring( 4, 6 )                 value “is”
      stateName.substring( 9, 11 )                value “pi”
```

What are **Arrays** ?

Arrays are data structures consisting of related data items all of the same type.

- An array type is a reference type. **Contiguous memory locations** are allocated for the array, beginning at the base address of the array.

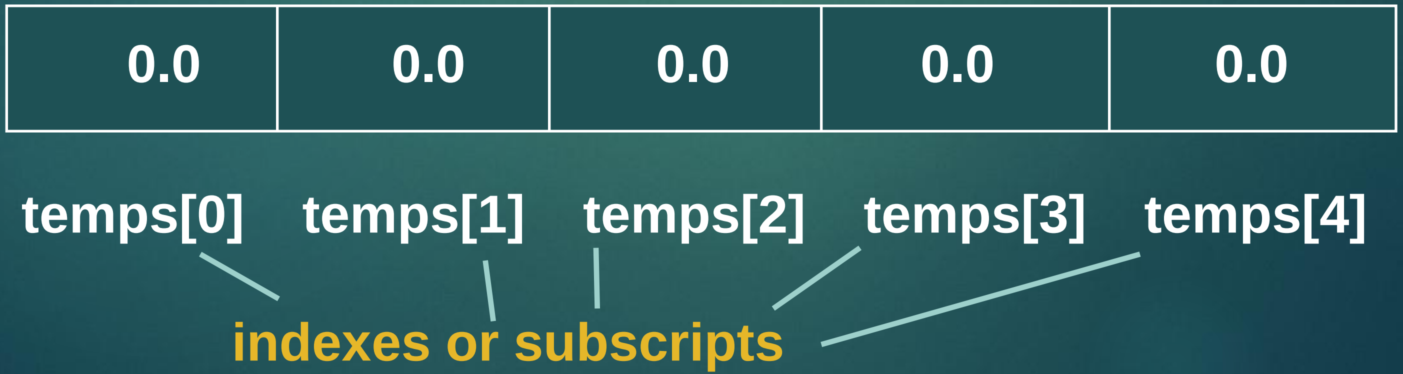
- A particular element in the array is accessed by using the array name together with the position of the desired element in square brackets. The position is called the **index or subscript**.

Example

- Declare and instantiate an array called `temps` to hold 5 individual double values.

number of elements in the array

```
double[ ]  temps = new double[ 5 ] ;
      // declares and  allocates memory
```



Using an **initializer list** in a declaration

```
int[ ]  ages  =  { 40, 13, 20, 19, 36 } ;


for  ( int  i = 0;  i < ages.length ; i++ )
       System.out.println( "ages[ " + i + " ] = " + ages[ i ] ) ;
```

```
ages[ 0 ] = 40
ages[ 1 ] = 13
ages[ 2 ] = 20
ages[ 3 ] = 19
ages[ 4 ] = 36
```

Passing **Arrays as Arguments**

- In Java an array is a reference type. What is passed to a method with an **array parameter is the address** of where the array object is stored.
- The **name of the array is actually a reference** to an object that contains the array elements and the public instance variable `length`.

```
public class AvgClass {
        public  static double  Avg( int[ ]  grades )  {   // grades is a parameter
                int  total = 0 ;
                for ( int i = 0 ;  i < grades.length ;  i++  )
                        total = total + grades[ i ] ;
                return  (double)  total / (double) grades.length ;
        }
        public static void main(String args[]) {
                int [] stuGrades = {55, 88, 44, 66, 77};
                System.out.println("Average=" + Avg(stuGrades) ); // stuGrades is an argument
        }
}
```

Declaration of **Two-Dimensional Array**

Array Declaration

```
DataType [ ][ ] ArrayName;
```

EXAMPLES:

```
double[][] alpha;

String[][] beta;

int[][] data;
```

Two-Dimensional Array Instantiation

Two-Dimensional Array Instantiation

ArrayName = n ew DataType [Expression1] [Expression2] ;

where each Expression has an integral value and specifies the number of components in that dimension

TWO FORMS FOR DECLARATION AND INSTANTIATION

```
    int[][] data;

    data = new int[6][12];
```

OR

```
    int[][] data = new int[6][12];
```

Indexes in Two-Dimensional Arrays

- Individual array elements are accessed by a pair of indexes. The first index represents the element's row, and the second index represents the element's column.

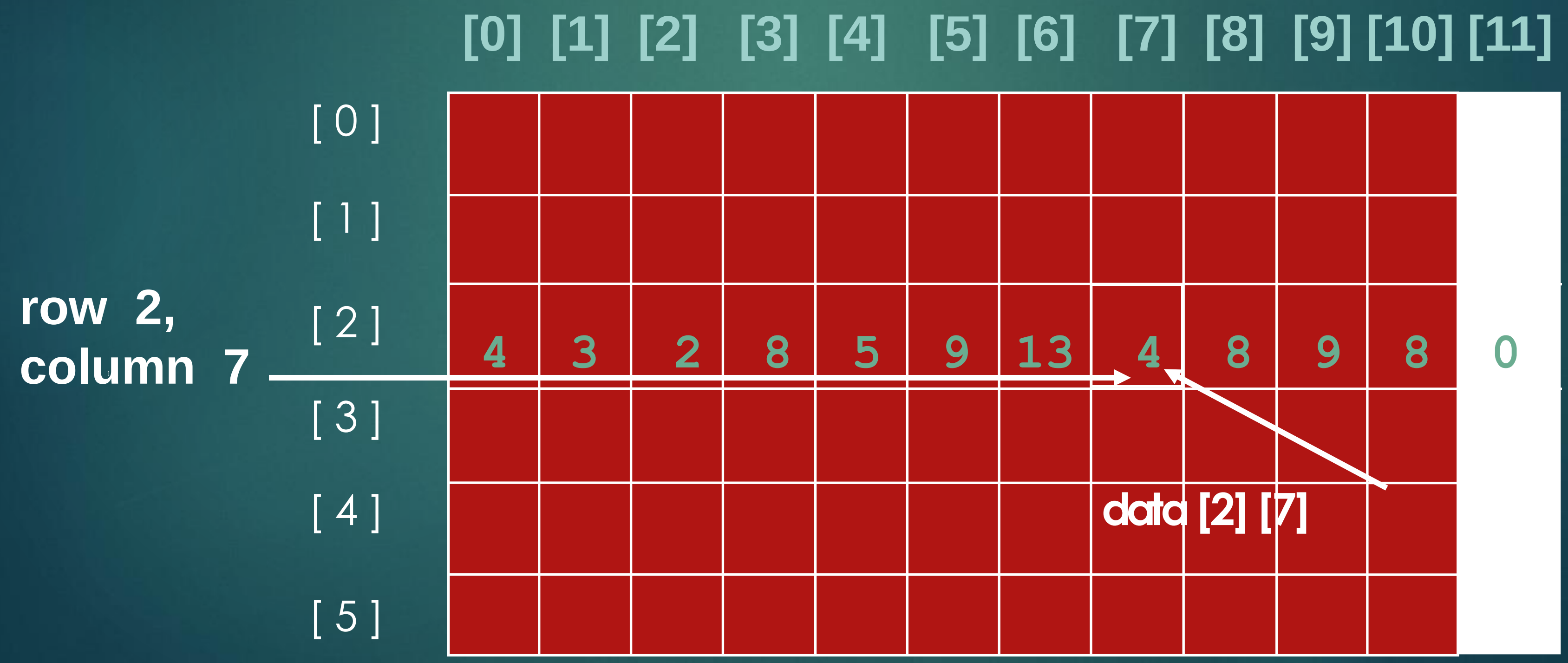
```
int[ ][ ] data;
data = new int[6][12] ;

data[2][7] = 4 ;      // row 2, column 7
```

Accessing an Individual Component

```
int [ ][ ] data;
data = new int [ 6 ] [ 12 ] ;

data [ 2 ] [ 7 ] = 4 ;
```



The `length` fields

```
int [ ] [ ] data = new int [ 6 ] [ 12 ] ;
```

data.length 6 gives the number of rows in array data

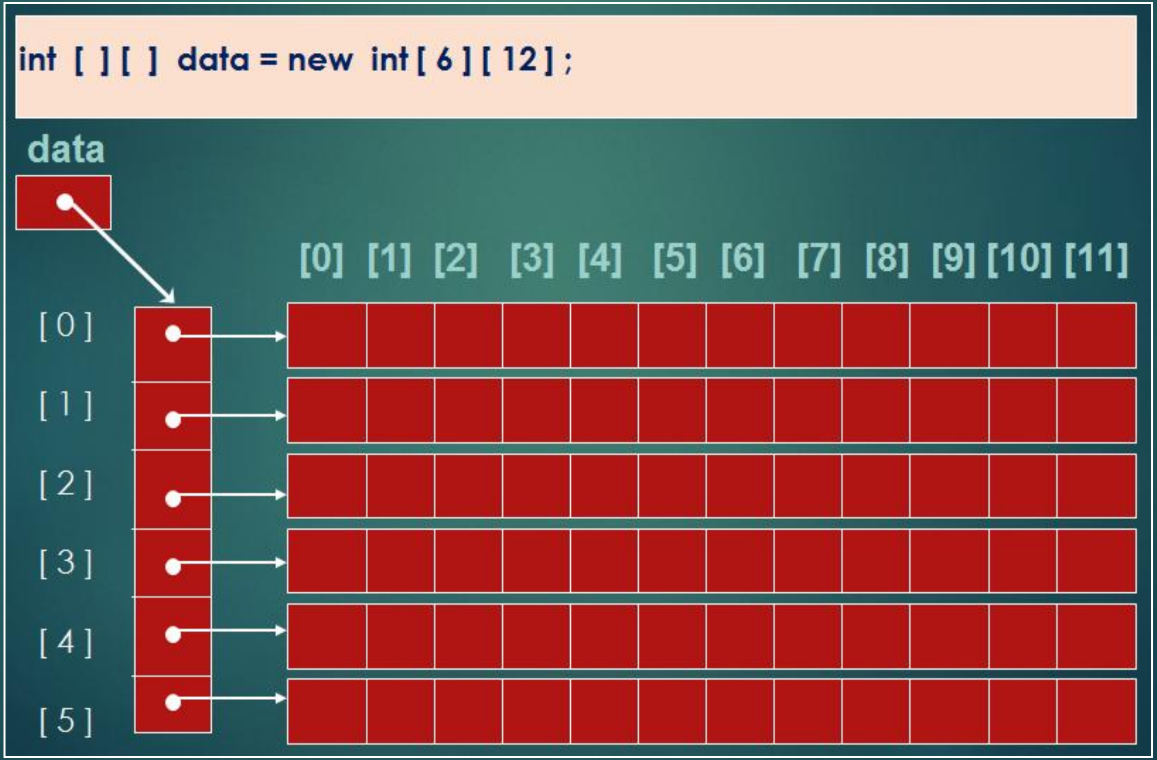
data [2]. length 12 gives the number of columns in row 2

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
[0]												
[1]												
[2]	4	3	2	8	5	9	13	4	8	9	8	0
[3]												
[4]												
[5]												

row 2

Java Implementation of 2D-Array

- In Java, actually, a two-dimensional array is itself a one-dimensional array of references to one-dimensional arrays.



```
int [ ][ ] data = new int[ 6 ][ 12 ];
```

Any questions ?