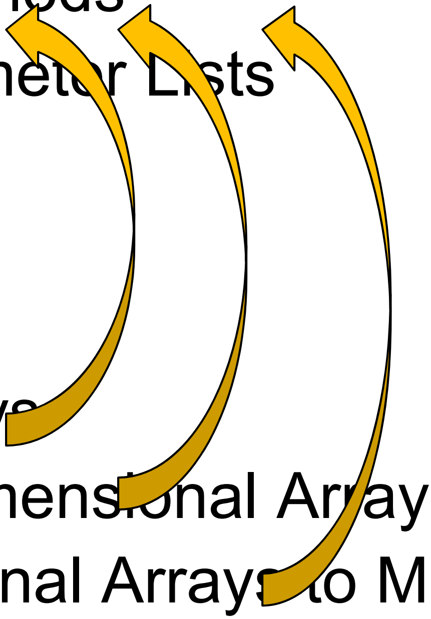
Course Syllabus

- Computer Systems & Java Programming
- Java Program Development
- Data and Operators
- Console Input/Output
- Branching
- Looping
- Methods

- Arrays
- Classes & Objects
- Strings & Characters
- Class Inheritance (Optional & Non-Examinable) – E-Learning
- Exception Handling
- File Input/Output

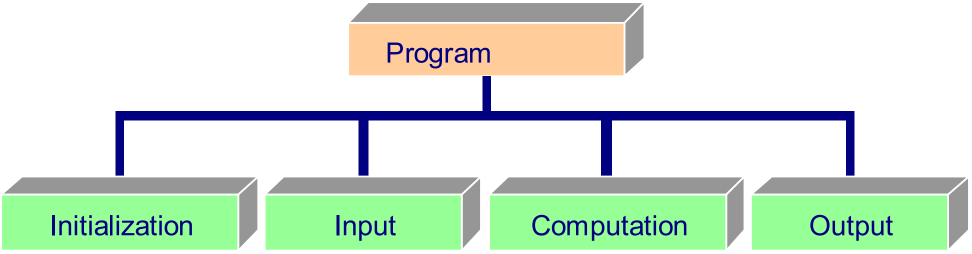
Chapter 9: Arrays

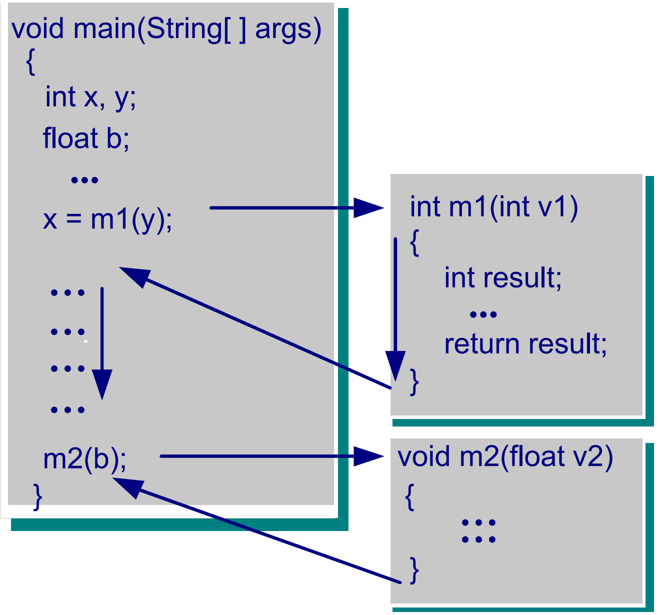
- **Why Learn & Use Arrays?**
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies



Review Chapter 8: Methods

- Method Definition
  - Method header
  - Method body
- Calling a Method
- Passing Parameters and Values
- Overloading Methods
- Scope of Variables
- Designing Programs with Methods
- Case Study





```
void main(String[ ] args)
 {
   int x, y;
   float b;
     •••
   x = m1(y);

   •••
   •••
   •••
   •••

   m2(b);
 }
```

```
int m1(int v1)
{
   int result;
     •••
   return result;
}
```

```
void m2(float v2)
{
     :::
}
```

Why Learn & Use Arrays?

- Arrays are a fundamental concept in most (if not all) programming languages like C, C++, Visual Basic, COBOL, etc.
- The idea is to have **a variable** that can be used to collect **a group** of ***data having the same type***

```
String student1 = “Tan Ah Meng” ;
String student2 = “Ahmad ..” ;
String student3 = “Sengar .. ” ;
String student4 = “Robert ..” ;
…….
……
String student400 = “Nguyen ..” ;
```

?

Sort by name ?

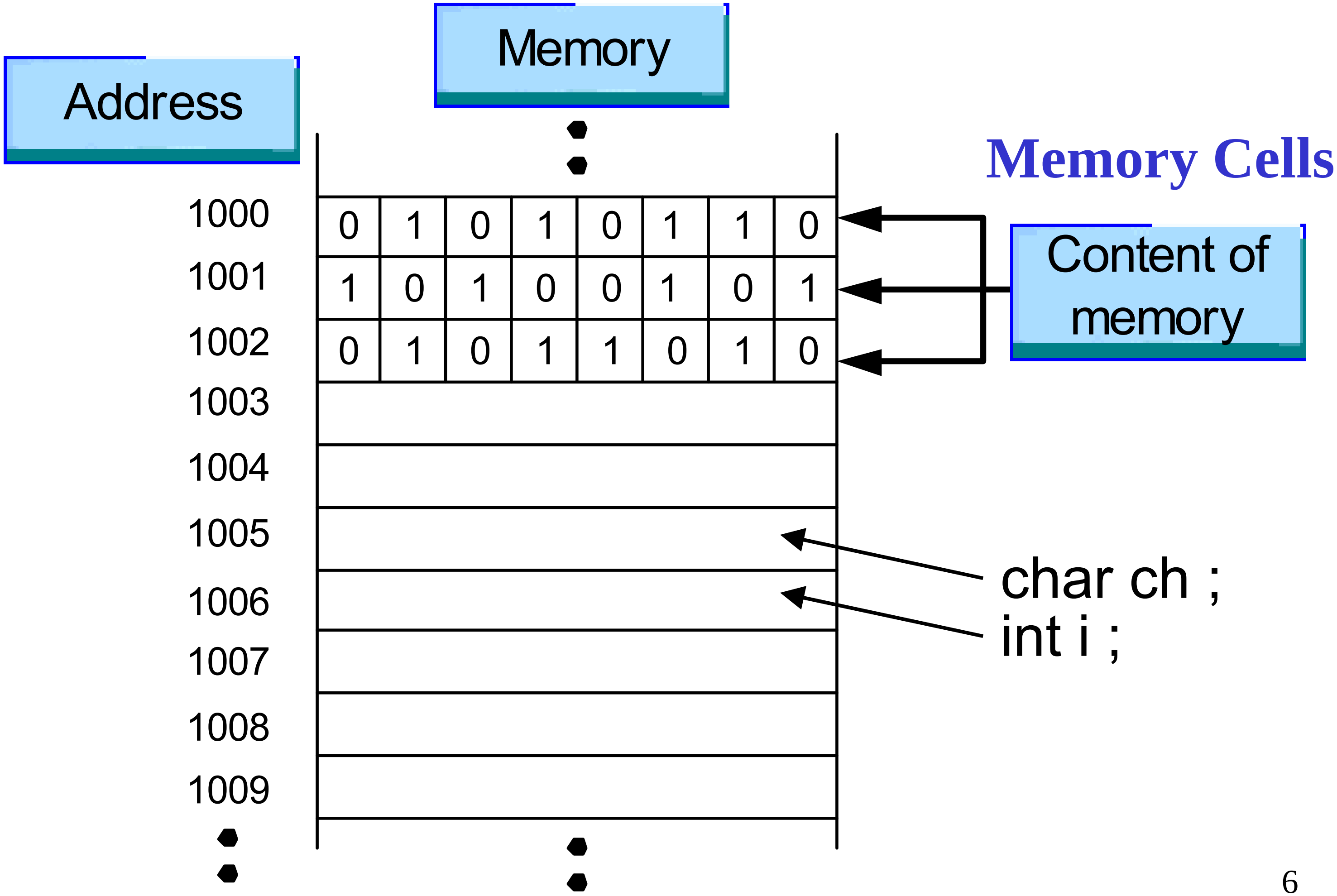
Find student by the name of …?

Chapter 9: Arrays

- Why Learn & Use Arrays?
- **Declaring and Creating Arrays**
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

FROM LECTURE 3

Computer Memory



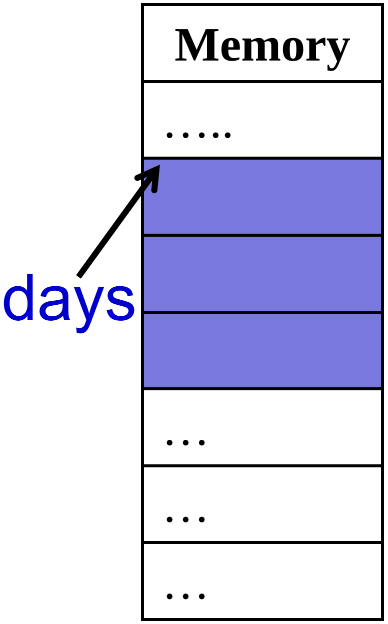
1000	0	1	0	1	0	1	1	0
1001	1	0	1	0	0	1	0	1
1002	0	1	0	1	1	0	1	0
1003								
1004								
1005								
1006								
1007								
1008								
1009								

Declaring Arrays

- An array is a series of elements having ***the same*** data type.
- The elements are stored ***sequentially*** in memory.
- **Declaring** arrays:

Format: Type [ ]*Variable* ;

e.g.: int [ ] days ;

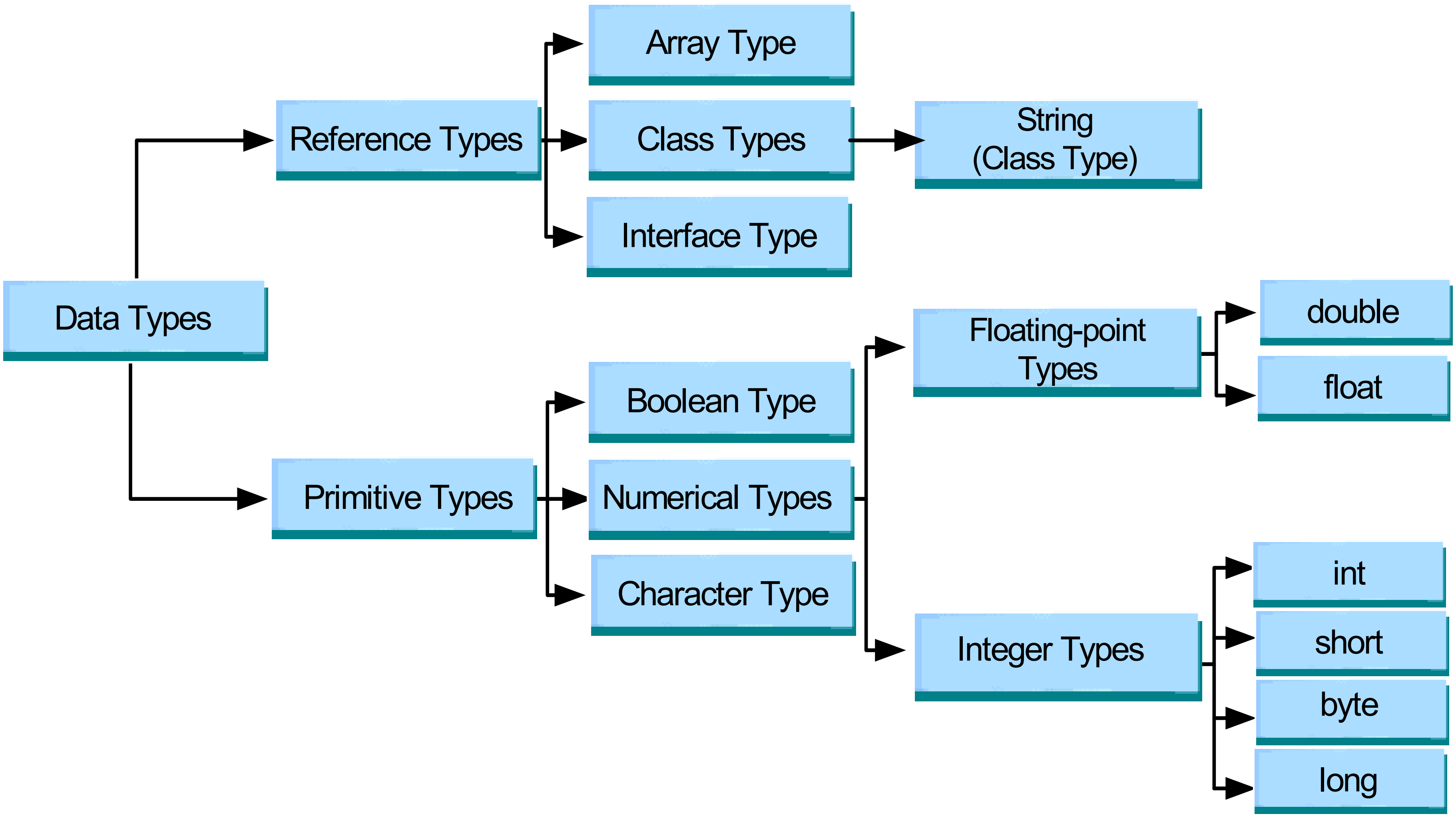


The empty brackets allow the Java compiler to know that the variable days is an array storing a sequence of integers.

days will be considered a reference. It will refer to some memory location where array starts.

FROM LECTURE 3

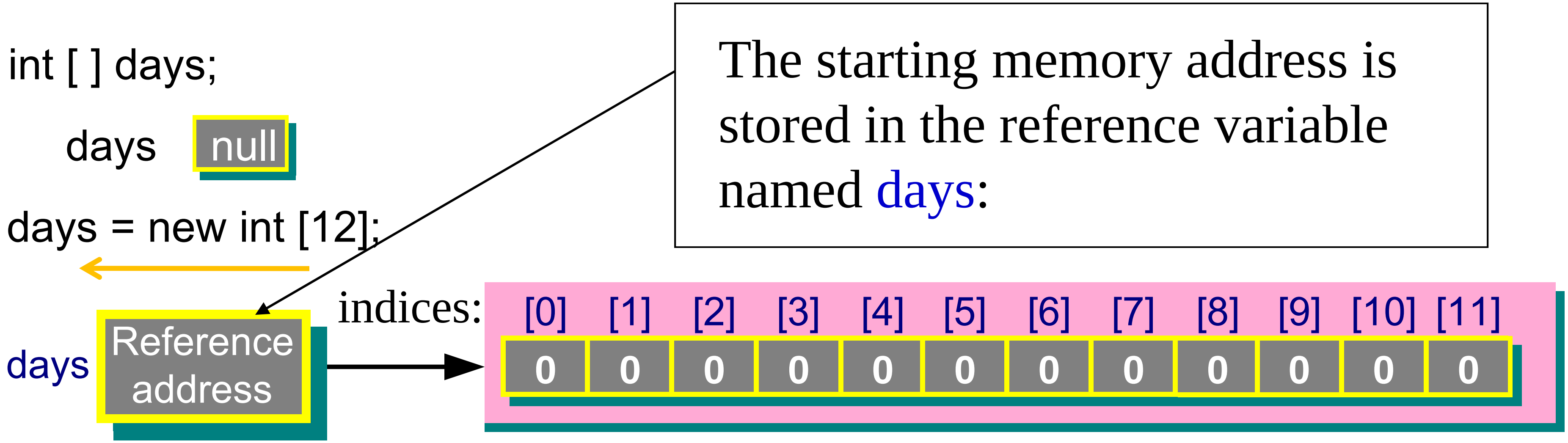
Data Types



Creating Arrays

- After declaring a reference, we can create (allocate memory) the array to assign to the reference.
- **Creating** arrays with 12 elements of integer type that stores the number of days in each month (initialized to 0):

```
int [ ] days ;   // declaring a reference
days = new int[12] ;  // allocate memory
```



int [ ] days;

days null

days = new int [12];

The starting memory address is stored in the reference variable named days:

days Reference address

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

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

- Alternatively, declaration and allocation statements can be combined in one statement:

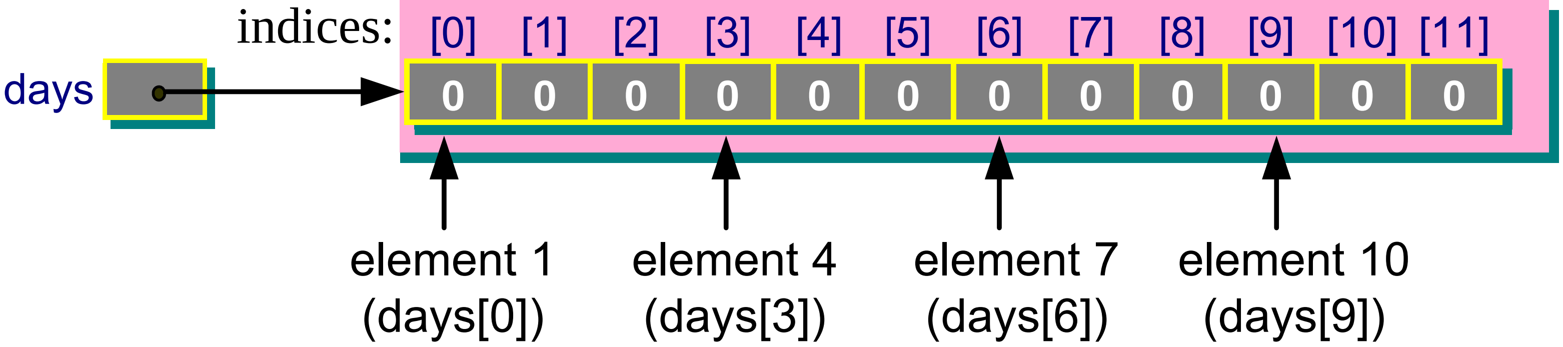
1. Declare the array 2. Create the array

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

or

```
static final int MAX_DAYS = 12;
int[ ] days = new int[ MAX_DAYS ];
```

- The starting memory address is stored in the reference variable named days:



element 1 (days[0])

element 4 (days[3])

element 7 (days[6])

element 10 (days[9])

- *indexed variables*: days[0], days[3], days[6], days[9]

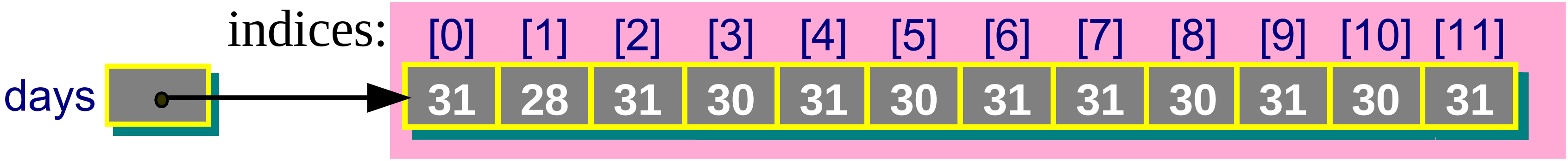
Initialization of Arrays

- **Assign** a value to an array element :

  e.g. days[0] = 31 ;
  days[1] = 28 ;
  …

- Another way: **Initialize** array variables at declaration using an initializer list that separates a list of initializers by commas.

  e.g. int[ ] days = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } ;



- The **array size** is determined by the number of initializers in the initializer list.

Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- **Operations on Arrays**
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Operations on Arrays

- Declaring an array variable sales:

```
double[ ] sales = new double[ 365 ] ;
```

- Accessing array elements:

```
Ex 1.   sales[0] = 143.50 ;

Ex 2.   if ( sales[23] == 50.0 ) ...

Ex 3.   sales[8] = sales[5] – sales[2] ;

Ex 4.   while ( sales[364] != 0.0 ) {...}

Ex 5.   for ( int i=0 ; i < 365 ; i++ )
              sales[ i ] = 0 ;
```

Subscripting an Array

IMPORTANT:

Arrays in Java (and C/C++) are indexed from 0 to SIZE-1.

When you try to address an array outside its range, you will get a run-time error.

For example:

```
char[ ] name = new char[ 12 ] ;
name[12] = 'c' ;             // run-time error will occur
for ( int i = 0 ; i <= 12 ; i++ )
      name[ i ] = 'x';       // error will occur when
                             // i=12 at run-time
```

Traversing Arrays

- Java stores the **size of the array** automatically in an instance variable named length.

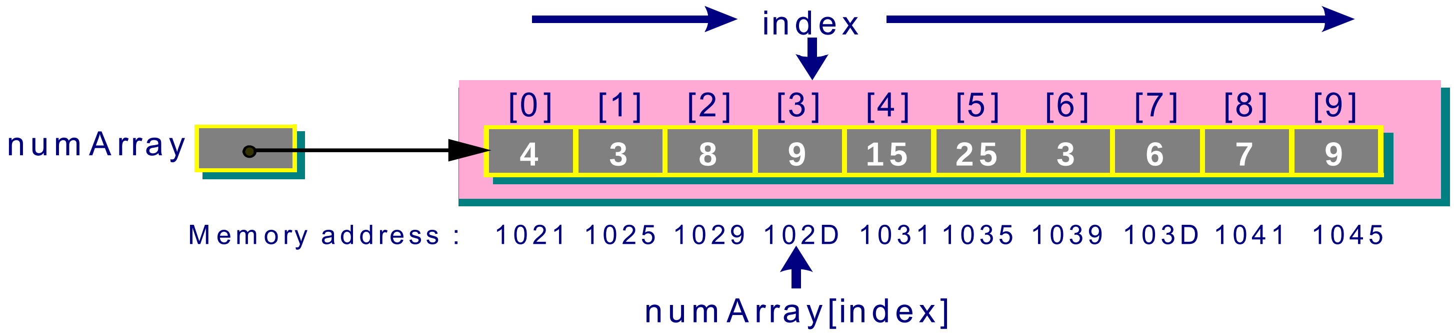
```
public class PrintingDays
{
  public static void main( String[] args )
  {
    int i;
    int[] days = {31,28,31,30,31,30,31,31,30,31,30,31};
    // print the number of days in each month
    for ( i=0 ; i < days.length ; i++ ) // traverse array
      System.out.println( "Month " + (i+1) +
        " has " +  days[i] + " days." );
  }
}
```

```
Program Output
Month 1 has 31 days.
Month 2 has 28 days.
...
Month 12 has 31 days.
```

Example: Finding Maximum Number

```
import java.util.Scanner ;
public class FindMaximum {
  public static void main
      (String[] args){
    final int MAX_INPUT = 10 ;
    int       index, max ;
    int[] numArray = new int[ MAX_INPUT ];
    Scanner sc = new Scanner( System.in );
    System.out.println( "Enter 10 numbers: " );
    for ( index = 0 ; index < MAX_INPUT ; index++ )
       numArray[index] = sc.nextInt() ;  // read input
    max = numArray[0];
    for ( index = 1 ; index < MAX_INPUT ; index++ )
       if ( max < numArray[ index ] )
            max = numArray[ index ] ;
    System.out.println( "The max value is " + max );
  }
}
```

```
Program Input and Output
Enter 10 numbers:
4 3 8 9 15 25 3 6 7 9
The max value is 25
```



Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- **Passing Arrays to Methods**
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Passing Arrays to Methods

- Two ways to pass arguments to methods:

  (1) **pass by value:** used for passing primitive data types
  (2) **pass by reference**: used for passing objects

- So, you may pass:
  (1) indexed variables as method arguments to a method.
  (2) In addition, an array is treated as an object and, hence, the entire array can be **passed by reference** to a method. i.e. the address of the first element of the array is passed to the method.

Example #1: Passing Indexed Variables

```
import java.util.Scanner;
public class PassingIndexedVariables {
    public static void main(String[ ] args) {
        final int MAX_INPUT=10;
        int index, max;
        int[] numArray = new int[MAX_INPUT];
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter 10 numbers: ");
        for (index = 0; index < MAX_INPUT; index++)
              numArray[index] = sc.nextInt();
        max = numArray[0];
        for (index = 1; index < MAX_INPUT; index++)
              max = larger(max, numArray[index] );
        System.out.println("The maximum value is " + max);
   }
   public static int larger(int first, int second)    {
        if (first > second)
              return first;
        else return second;
   }
}
```

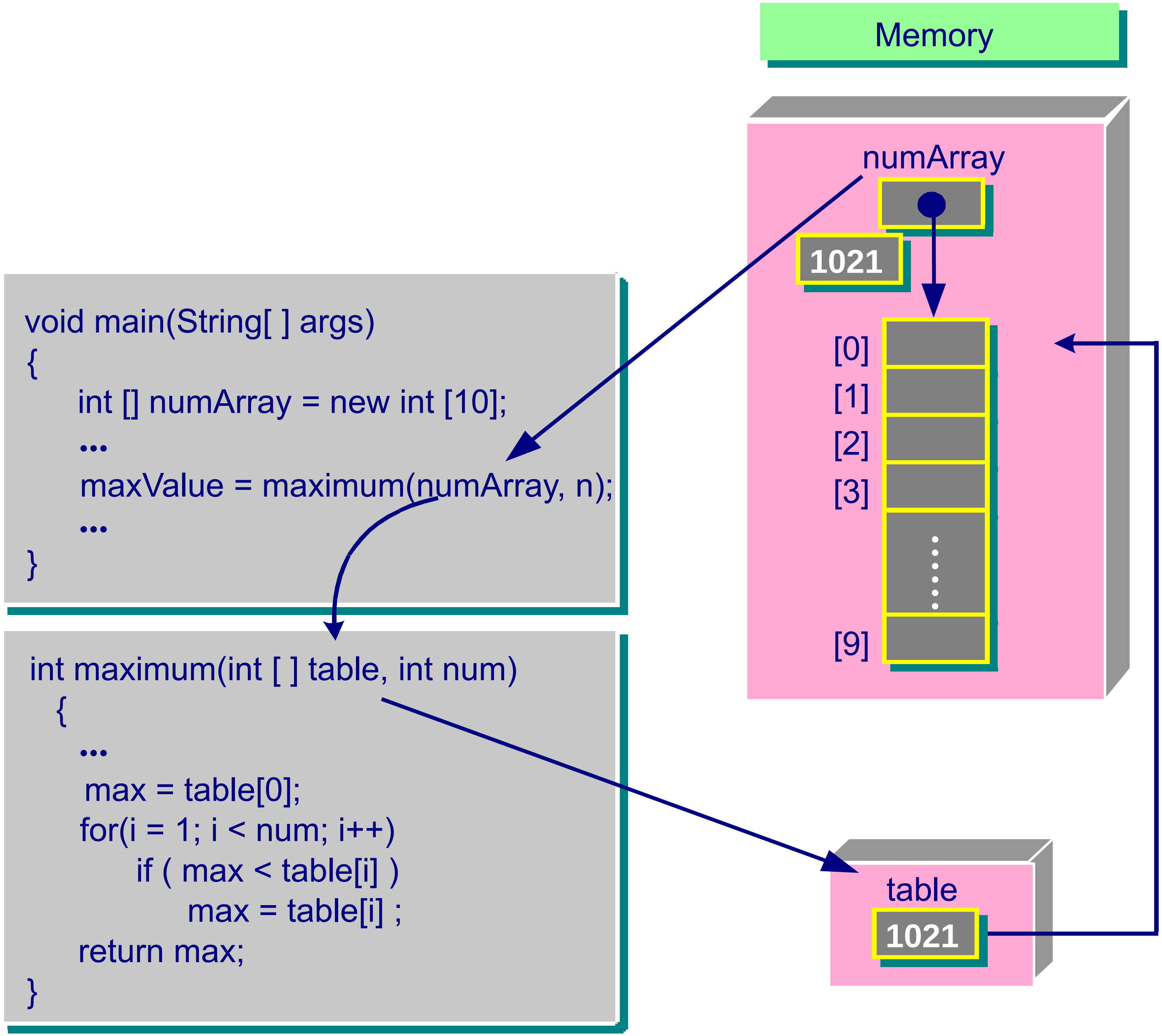
Example #2: Passing Entire Array

```
import java.util.Scanner;
public class PassingArrays {
    public static void main(String[ ] args) {
        int maxValue, index, n;
        int[] numArray = new int[10];
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of values: ");   n = sc.nextInt();
        System.out.print("Enter the values: ");
        for (index = 0; index < n; index++)
              numArray[index] = sc.nextInt();
        maxValue = maximum(numArray, n);
        System.out.println("The maximum value is " + maxValue);
   }
   public static int maximum(int[ ] table, int num) {
        int i, max;
        max = table[0];
        for (i = 1; i < num; i++)
              if ( max < table[i] )
                   max = table[i] ;
        return max;
    } }
```

Program Input and Output

```
Enter the number of values: 5
Enter the values: 12 5 13 20 8
The maximum value is 20
```

Passing Entire Array as Method Arguments



```
void main(String[ ] args)
{
      int [] numArray = new int [10];
      ...
      maxValue = maximum(numArray, n);
      ...
}
```

```
int maximum(int [ ] table, int num)
   {
      ...
       max = table[0];
      for(i = 1; i < num; i++)
             if ( max < table[i] )
                   max = table[i] ;
      return max;
}
```

Java doesn’t pass the actual array, only the reference to its memory location.

Example #3: Return An Array from the Method

```
import java.util.Scanner;
public class ReturningArrays {
  public static void main(String[] args) {
    int index, n;
    int[] numArray = new int[10];
    Scanner sc = new Scanner(System.in);
    System.out.println("Enter 10 numbers: ");
    for (index = 0; index < 10; index++)
       numArray[index] = sc.nextInt();
    numArray = multiplyArray(numArray);
    System.out.println("The multiplied values are: ");
    for (index = 0; index < 10; index++)
      System.out.print(numArray[index] + " "); }
  public static int[] multiplyArray(int[] table) {
    int i; int[] temp = new int[table.length];
    for ( i = 0 ; i < temp.length ; i++ )
       temp[i] = table[i] * 5 ;
    return temp;
  }
}
```

```
Program Input and Output
Enter 10 numbers:
12 5 13 20 8 16 9 40 30 25
The multiplied values are:
60 25 65 100 40 80 45 200 150 125
```

Pass/Return by Memory reference

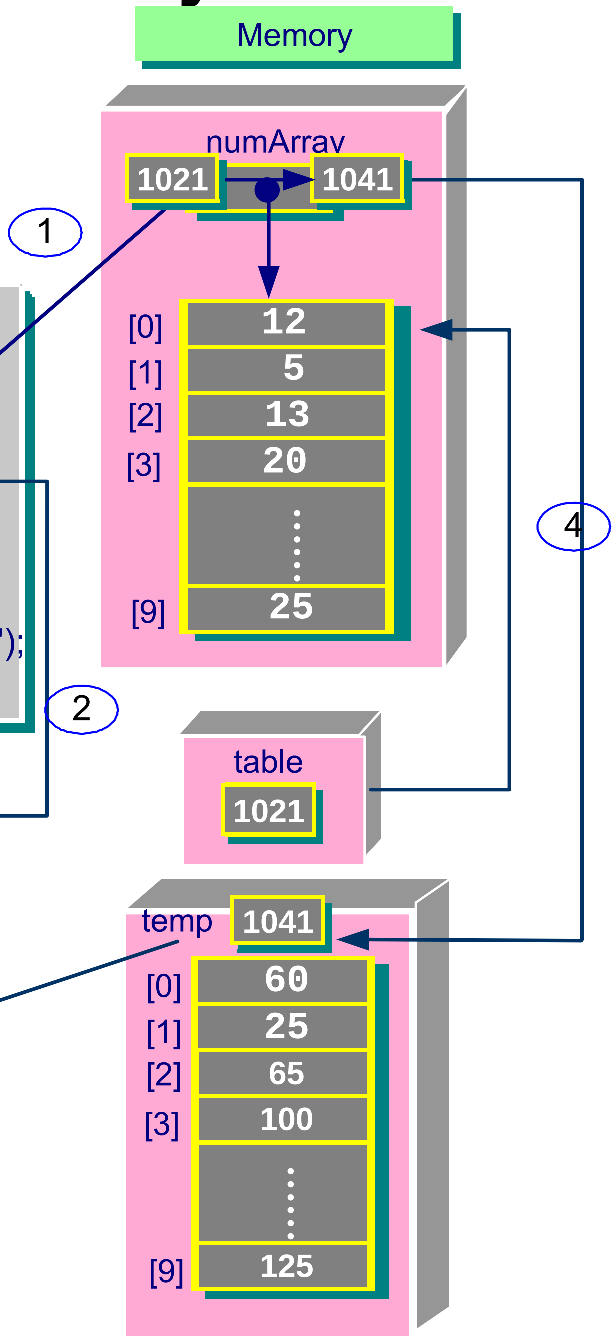
In this example, an array is returned by the method “multiplyArray(..)”

We say it returns “an array”, but actually the return is the memory reference to the newly created array.

```
void main(String[] args){
    int index, n;
    int[] numArray = new int[10];
     . . .
    numArray = multiplyArray(numArray);
    System.out.println("The multiplied
       values are: ");
       for (index = 0; index < 10; index++)
          System.out.print(numArray[index] + " ");
}
```

```
int[] multiplyArray(int[] table)
{
    int i;
    int[] temp = new int[table.length];

    for (i = 0; i < temp.length; i++)
        temp[i] = table[i] * 5;
     return temp;
}
```



Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- **Variable Length Parameter Lists**
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Variable Length Parameter Lists

- A new feature in **Java 2 Version 5.0**!

- Example:

```
public static int maximum( int … table )
```

The ellipsis (…) is used to indicate that the method can accept **variable number of parameters**.

Here, the variable **table** becomes an array.

```
public class VarLengthParameterList
{
   public static void main(String[] args)
   {
     int max;
     max = maximum(1,2,3);
     System.out.println("The maximum value is " + max);
     max = maximum(1,3,5,43,9,11,15);
     System.out.println("The maximum value is " + max);
   }
   public static int maximum( int ... table )
   {
     int i, temp=0;
     if ( table.length != 0 )
     {
        for (int num: table)  // enhanced for loop!!!
            if (num > temp)
                  temp = num;
     }
     return temp ;
   }
}
```

Program Output

```
The maximum value is 3
The maximum value is 43
```

Chapter 9: Arrays

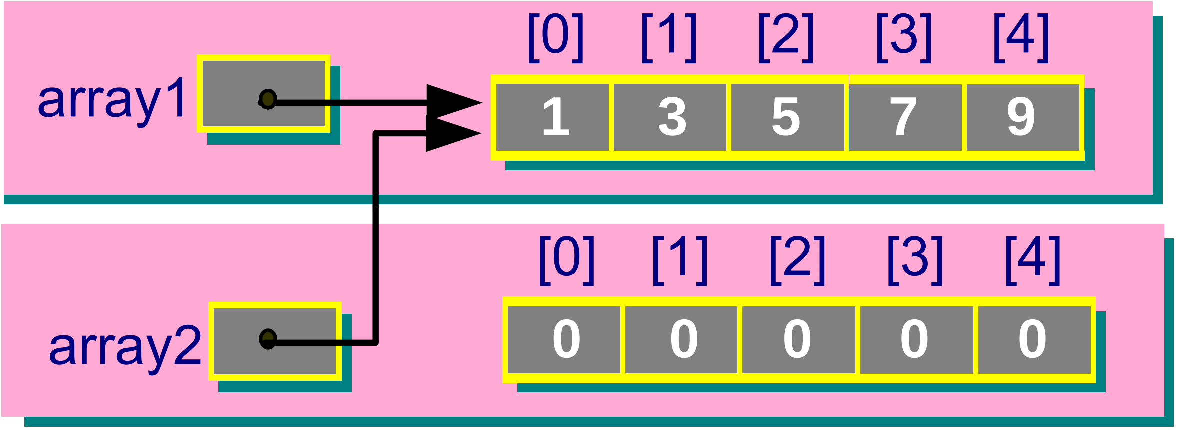
- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- **Copying Arrays**
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Copying Arrays

- Example:

```
int [ ] array1 = { 1 , 3 , 5 , 7 , 9 } ;
int [ ] array2 = new int[5]  ;
```

Copying an array: array2 = array1???



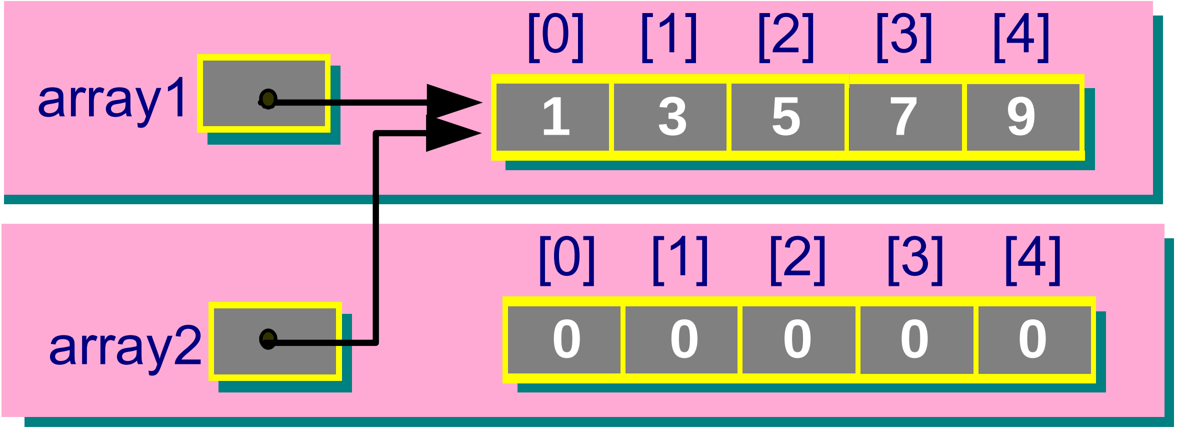
Copying Arrays

- Example:

  int [ ] array1 = { 1 , 3 , 5 , 7 , 9 };
  int [ ] array2 = new int[5];

  Copying an array: array2 = array1???

  Java does not automatically copy arrays for you, instead the above statement will have two variables referencing the same array! So be careful!



```java
public class CopyingArrays
{
  public static void main(String[] args)
  {
    int index;
    int[] array1 = {1, 3, 5, 7, 9};
    int[] array2 = new int[5];
    for (index = 0; index < 5; index++)
      array2[index] = array1[index];
    System.out.print("Array 1: ");
    printArray(array1);
    System.out.print("Array 2: ");
    printArray(array2);
  }
  public static void printArray(int[] table)
  {
    for (int i = 0; i < table.length; i++)
      System.out.print(table[i] + " ");
    System.out.println();
  }
}
```

Explicitly copy an array, element by element.

Program Output

```
Array 1: 1 3 5 7 9
Array 2: 1 3 5 7 9
```

Copying Arrays

- Another way of copying arrays is to use the static arraycopy method in System class:

```
System.arraycopy(
        Source_Array            ,
        Source_Position         ,
        Destination_Array       ,
        Destination_Position    ,
        Length
);
```

Examples:

```
System.arraycopy( array1, 0, array2, 0, array1.length );
System.arraycopy( array1, 0, array2, 1, 4 );
        - array1[0] -> array2[1]; … ; array1[3] -> array2[4]
```

Chapter 9: Arrays

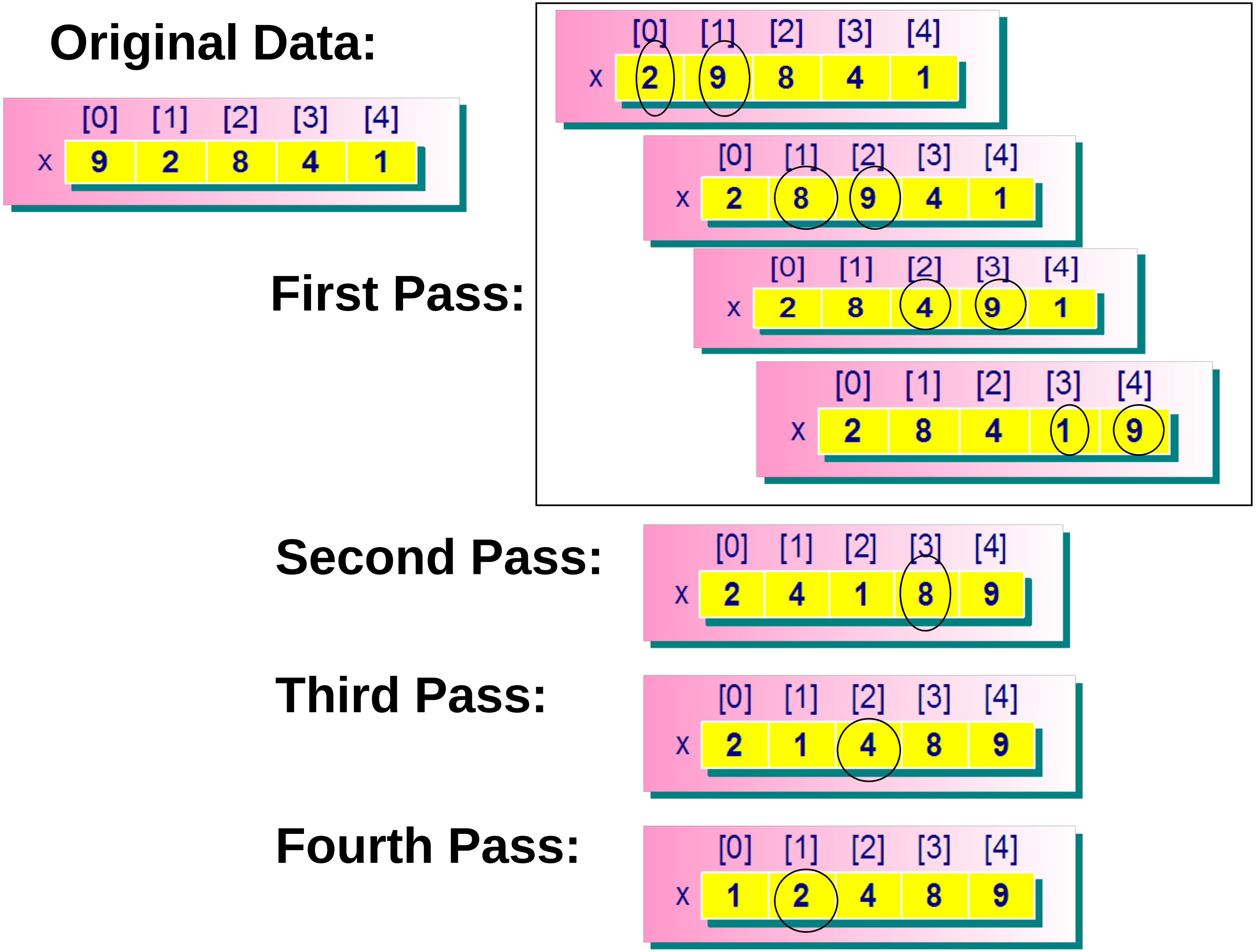
- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- **Sorting Arrays**
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Sorting Arrays

Many sorting algorithms: bubble sort, merge sort, quick sort, etc.

Bubble sort (the simplest) operators on an array:

1. **Compare** the first two numbers: if the second number is smaller than the first number, swap the two numbers.
2. **Move down** one number (**consider the next two numbers: 2nd and 3rd**), **compare** them and **swap** them if necessary (same rule as before).
3. **Repeat compare** and **swap** until finish processing the last two numbers in the array. One sequence of comparing and swapping is called one **pass**.
4. **Repeat** N-1 passes OR **stop** when no swap happened in previous pass (N = number of elements).



Original Data:

	[0]	[1]	[2]	[3]	[4]
x	9	2	8	4	1

First Pass:

	[0]	[1]	[2]	[3]	[4]
x	2	9	8	4	1

	[0]	[1]	[2]	[3]	[4]
x	2	8	9	4	1

	[0]	[1]	[2]	[3]	[4]
x	2	8	4	9	1

	[0]	[1]	[2]	[3]	[4]
x	2	8	4	1	9

Second Pass:

	[0]	[1]	[2]	[3]	[4]
x	2	4	1	8	9

Third Pass:

	[0]	[1]	[2]	[3]	[4]
x	2	1	4	8	9

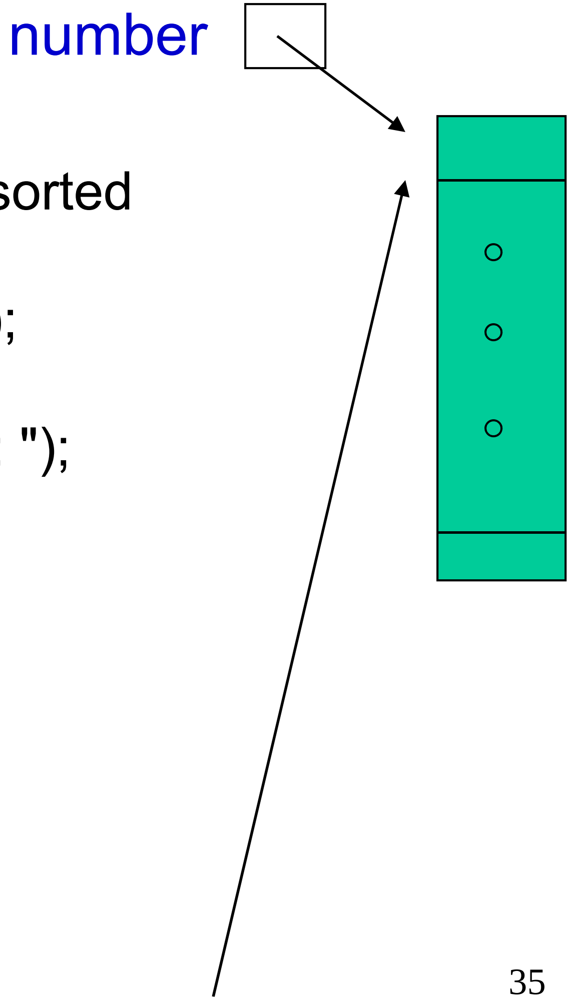
Fourth Pass:

	[0]	[1]	[2]	[3]	[4]
x	1	2	4	8	9

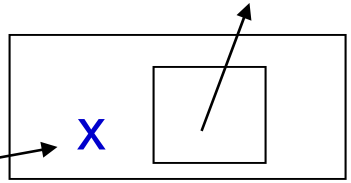
Result: in sorted order

Example: Bubble Sort Algorithm

```
import java.util.Scanner;
public class BubbleSortApp
{
  public static void main(String[ ] args)
  {
    int i, n;
    int[ ] number = new int[10];   // array to be sorted
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter number of items: ");
    n = sc.nextInt();
    System.out.print("Enter the list of numbers: ");
    for (i = 0; i < n; i++)
          number[i] = sc.nextInt();
    bubbleSort( number , n );
    System.out.print("The sorted array is: ");
    for (i = 0; i < n; i++)
          System.out.println(number[i] + " ");
   }
```



Example: Bubble Sort Algorithm



```
public static void bubbleSort(int[] x, int n) {
   int tempValue, pass, index;
   for (pass = 0; pass < n-1; pass++) {                  // n-1 passes
        for (index=0; index < n-1-pass; index++) { // for each pass
           if (x[index] > x[index+1])  {                 // comparison
               tempValue = x[index];                     // swap process
               x[index] = x[index+1];
               x[index+1] = tempValue;
           }
        }
   }
 }
}
```

```
Program Input and Output
Enter number of items: 6
Enter the list of numbers: 6 2 7 9 1 4
The sorted array is: 1 2 4 6 7 9
```

Chapter 9: Arrays

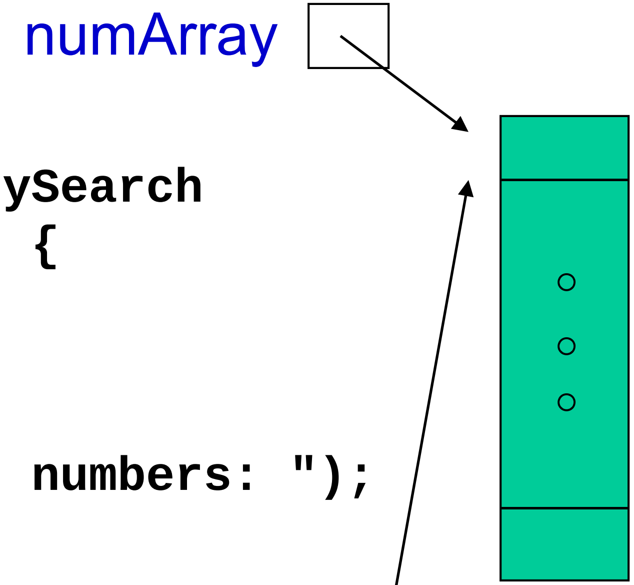
- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- **Searching Arrays**
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Searching Arrays

Two searching algorithms:

- **Linear search**
  - It compares each element in the array with the search key until a match is found or the end of the array is reached.
- **Binary search (Optional)**
  - It is used for **large** and **sorted** arrays.
  - It first locates the **middle** element in the array and compares it with the search key.
  - If matched, the search key is found, and the index of the array is returned.
  - If not, we repeat the process by searching **one half** of the array.

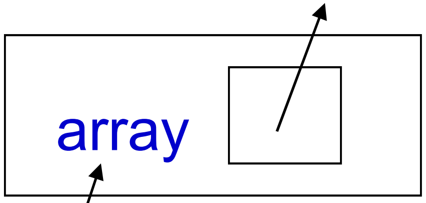
Example: Search Algorithms



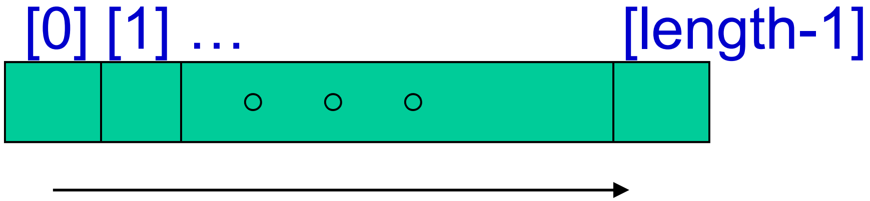
```
import java.util.Scanner;
public class LinearSearch {  // Or BinarySearch
  public static void main(String[] args) {
    int i, searchkey, found;
    int[] numArray = new int[10];
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter a list of 10 numbers: ");
    for (i = 0; i < 10; i++)
       numArray[i] = sc.nextInt();
    System.out.print("Enter key to be searched: ");
    searchkey = sc.nextInt();
    found = linearSearch(numArray, searchkey);
    // found = BinarySearch(numArray, searchkey);

    if (found != -1)
      System.out.println("Search value found at: " + found);
    else
      System.out.println("Search value not found.");
  }
```

Example: Linear Search Algorithm

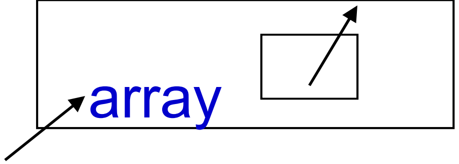


```
  public static int linearSearch(int[] array, int key){
    int index;
    for (index = 0; index < array.length; index++)
      if (array[index] == key)
         return index;
    return -1;
  }
}
```

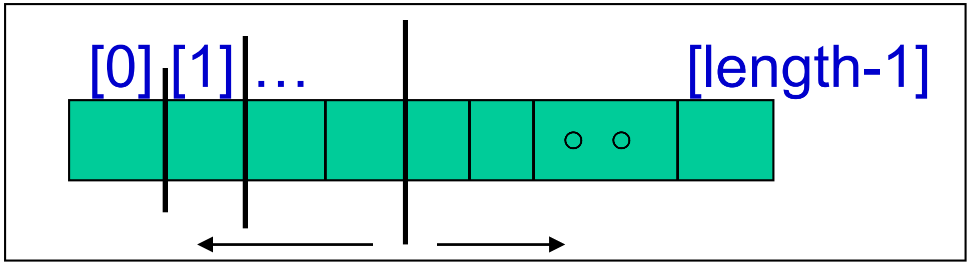


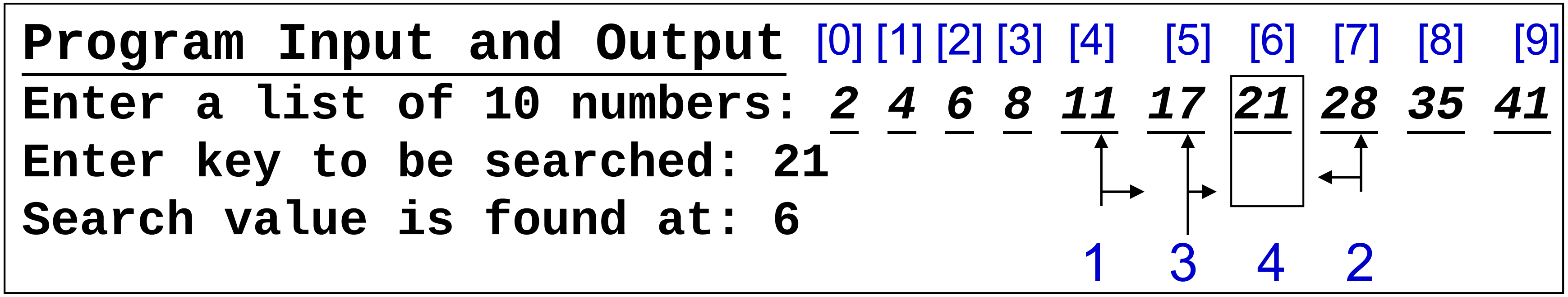
```
Program Input and Output
Enter a list of 10 numbers: 5 1 8 9 3 42 7 0 14 21
Enter key to be searched: 9
Search value found at: 3
```

Example: Binary Search Algorithm



```
public static int binarySearch(int[] array, int key) {
  int middle;                          // mid-point
  int first = 0;                       // first position in array
  int last = array.length-1;           // last position in array
  while (first <= last) {
    middle = (first + last)/2;
    if (key == array[middle])          // search key found
       return middle;
    else if (key < array[middle])
       last = middle – 1;
    else
       first = middle + 1;
  }
  return –1; // not found
 }
}
```





```
Program Input and Output
Enter a list of 10 numbers: 2 4 6 8 11 17 21 28 35 41
Enter key to be searched: 21
Search value is found at: 6
```

Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- **Multidimensional Arrays**
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- Case Studies

Multidimensional Arrays

- **Declared** as consecutive pairs of brackets, and **create** using the **new** operator.

The syntax:

```
int[ ][ ] x = new int[3][5];     // row -> column
```

i.e.

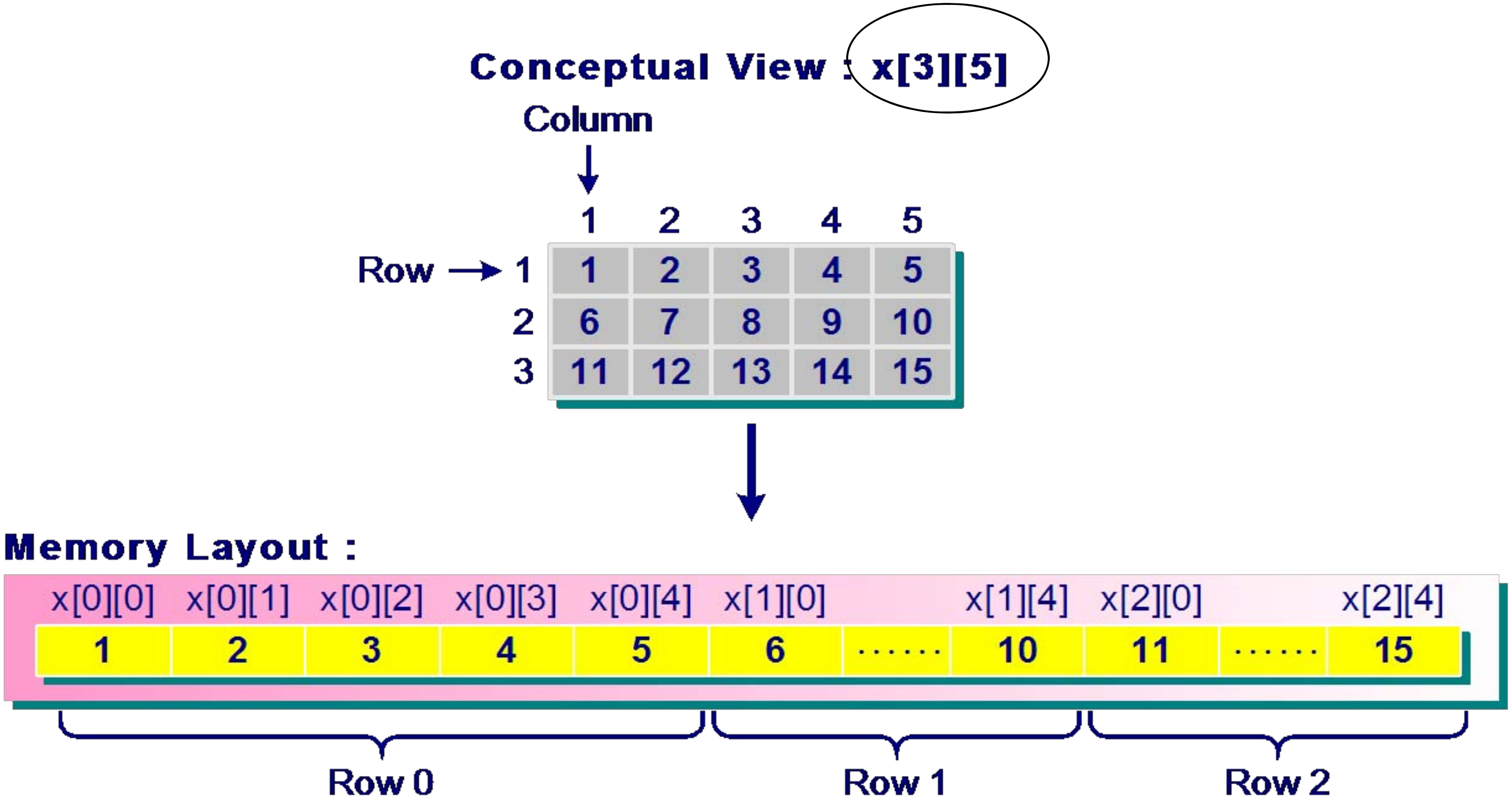
```
int[ ][ ] x;
x = new int[3][5];
```

e.g.

```
char[ ][ ][ ] x = new char[3][4][5];  // page -> row -> column
```

Stored in memory in row-major order.

	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]
Row 1	x[1][0]	x[1][1]	x[1][2]	x[1][3]	x[1][4]
Row 2	x[2][0]	x[2][1]	x[2][2]	x[2][3]	x[2][4]



Conceptual View : x[3][5]

Column

Row

	1	2	3	4	5
1	1	2	3	4	5
2	6	7	8	9	10
3	11	12	13	14	15

Memory Layout :

x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]	x[1][0]		x[1][4]	x[2][0]		x[2][4]
1	2	3	4	5	6	......	10	11	......	15

Row 0 Row 1 Row 2

- **Initializing** multidimensional arrays (using **initializer**): enclose each row in braces.

```
int[ ][ ] x =     {{ 1, 2},          /* 1st row */
                   { 6, 7} };        /* 2nd row */
```

Equivalent to

```
int[ ][ ] x = new int[2][2];
x[0][0] = 1;            x[0][1] = 2;
x[1][0] = 6;            x[1][1] = 7;
```

- For higher dimensional arrays:

```
int [ ][ ][ ] array = {
                           { {1,1},{0,0},{1,1} },
                            { {0,0},{1,2},{0,1} }
                      };
```

gives **a[2][3][2]** dimensioned array.

Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- **Operations on Multidimensional Arrays**
- Passing Multidimensional Arrays to Methods
- Case Studies

Operations on Multidimensional Arrays

```
// traversing a 2-dimensional array
public class MultiDimensionalArray {
  public static void main(String[] args) {
  int[][] array = {
      {  5 , 10 , 15 },
      { 10 , 20 , 30 },
      { 20 , 40 , 60 }
  };
  int row, column, sum ;

  // sum of rows
  for ( row = 0 ; row < 3 ; row++ )
  {
    sum = 0 ;
    for ( column = 0 ; column < 3 ; column++ )
      sum += array[row][column] ;
    System.out.println( "Sum of elements in row "
      + row + " is " + sum );
  }
  // sum of columns
  for (column = 0; column < 3; column++) {
      sum = 0;
      for (row = 0; row < 3; row++)
         sum += array[row][column];
      System.out.println(“Sum of elements in column "
         + column + " is " + sum);
  }
 }
}
```

Program Output

```
Sum of elements in row 0 is 30
Sum of elements in row 1 is 60
Sum of elements in row 2 is 120
Sum of elements in column 0 is 35
Sum of elements in column 1 is 70
Sum of elements in column 2 is 105
```

Example: Traversing 2-D Arrays Using length

```
public class MultiDimensionalArray {
  public static void main(String[] args) {
  int[][] array = {
    {5, 10, 15},
    {10, 20, 30},
    {20, 40, 60}
   };
  int row, column, sum;

  // sum of rows
  for (row = 0; row < array.length; row++) {
    sum = 0;
    for (column = 0; column< array[row].length; column++)
      sum += array[row][column];
    System.out.println(“Sum of elements in row “
      + row + " is " + sum);
  }
  ….
}
```

The array is actually a 1-dimensional array of length 3 and each indexed variable is also an 1-dimensional array of length 3 of element type int.

Another Application: Matrix

Matrix - typically a 2D array of elements (numerical values)

Example: Matrix Addition

$$\begin{bmatrix}1 & 3 & 1\\ 1 & 0 & 0\\ 1 & 2 & 2\end{bmatrix} + \begin{bmatrix}0 & 0 & 5\\ 7 & 5 & 0\\ 2 & 1 & 1\end{bmatrix} = \begin{bmatrix}1+0 & 3+0 & 1+5\\ 1+7 & 0+5 & 0+0\\ 1+2 & 2+1 & 2+1\end{bmatrix} = \begin{bmatrix}1 & 3 & 6\\ 8 & 5 & 0\\ 3 & 3 & 3\end{bmatrix}.$$

Example: Matrix Multiplication

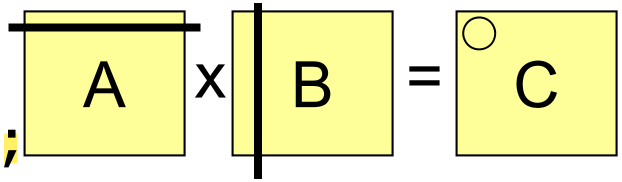
$$\begin{bmatrix}\overset{00}{1} & \overset{01}{0} & \overset{02}{2}\\ \underset{10}{-1} & \underset{11}{3} & \underset{12}{1}\end{bmatrix} \times \begin{bmatrix}\overset{00}{3} & \overset{01}{1}\\ \overset{10}{2} & \overset{11}{1}\\ \overset{20}{1} & \overset{21}{0}\end{bmatrix} = \begin{bmatrix}(1\times 3+0\times 2+2\times 1) & (1\times 1+0\times 1+2\times 0)\\ (-1\times 3+3\times 2+1\times 1) & (-1\times 1+3\times 1+1\times 0)\end{bmatrix}$$

$$= \begin{bmatrix}\overset{00}{5} & \overset{01}{1}\\ \underset{10}{4} & \underset{11}{2}\end{bmatrix}.$$

http://en.wikipedia.org/wiki/Matrix_(mathematics)
http://mathworld.wolfram.com/MatrixMultiplication.html
http://www.mathresource.iitb.ac.in/linear%20algebra/example2.0.1/index.html

Example: Matrix Multiplication

```
public class MatrixMultiApp { // for 3x3 matrices
  public static void main(String[] args) {
    int[][] A = { {1, 2, 3},{2, 3, -1},{3, -1, 2}};
    int[][] B = { {1, 2, 3},{5, 7, 9},{9, 11, 13}};
    int[][] C = new int[3][3];
    int l, m, n;
    for (l = 0; l < 3; l++) { // matrix multiplication
      for (m = 0; m < 3; m++) {
        C[l][m] = 0;
        for (n = 0; n < 3; n++)
            C[l][m] += A[l][n]*B[n][m];
      }
    }
    System.out.println("The product is: ");
    for (l = 0; l < C.length; l++) { // print matrix
      for (m = 0; m < C[l].length; m++)
        System.out.print(C[l][m] + " ");
      System.out.println();
    }
  }
}
```



Program Output

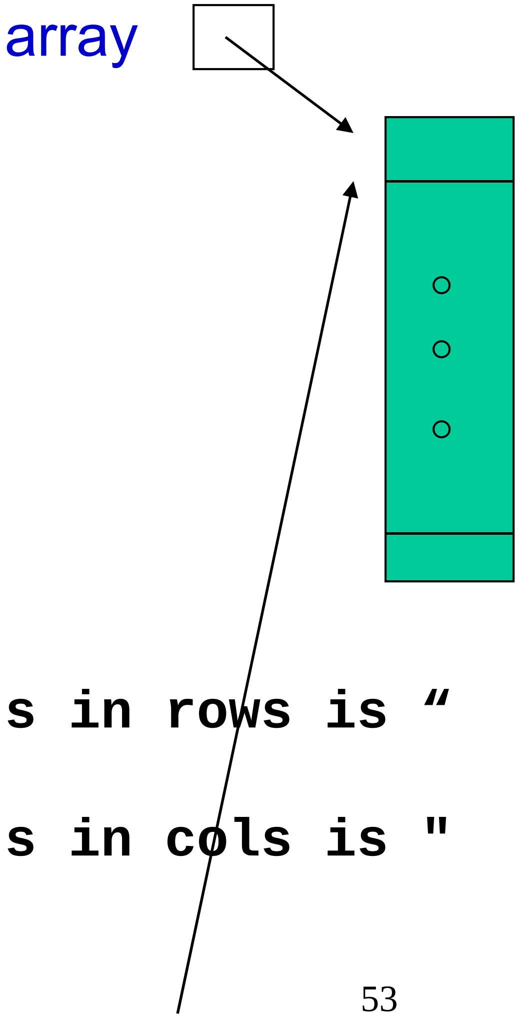
```
The product is:
38 49 60
8 14 20
16 21 26
```

Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- **Passing Multidimensional Arrays to Methods**
- Case Studies

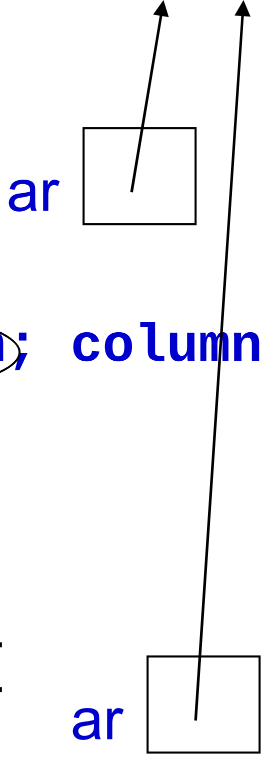
Passing 2-dimenional Arrays as Arguments

Example 1: Sum of rows and sum of columns



```
public class MultiDiArraysApp {
  public static void main(String[] args){
    int[][] array = {
      {5, 10, 15},
      {10, 20, 30},
      {20, 40, 60}
     };
    int totalRow, totalColumn;
    totalRow    = sumOfRows(    array );
    totalColumn = sumOfColumns( array );
    System.out.println("Sum of all elements in rows is “
      + totalRow);
    System.out.println("Sum of all elements in cols is "
      + totalColumn);
  }
```

```
 public static int sumOfRows(int[][] ar) {
    int row, column;
    int sum=0;
    for (row = 0; row < ar.length; row++) {
       for (column = 0; column < ar[row].length; column++)
             sum += ar[row][column];
    }
    return sum;
 }
 public static int sumOfColumns(int[][] ar) {
    int row, column;
    int sum=0;
    for (column = 0; column < ar[0].length; column++) {
       for (row = 0; row < ar.length; row++)
             sum += ar[row][column];
    }
    return sum;
 }
}
```

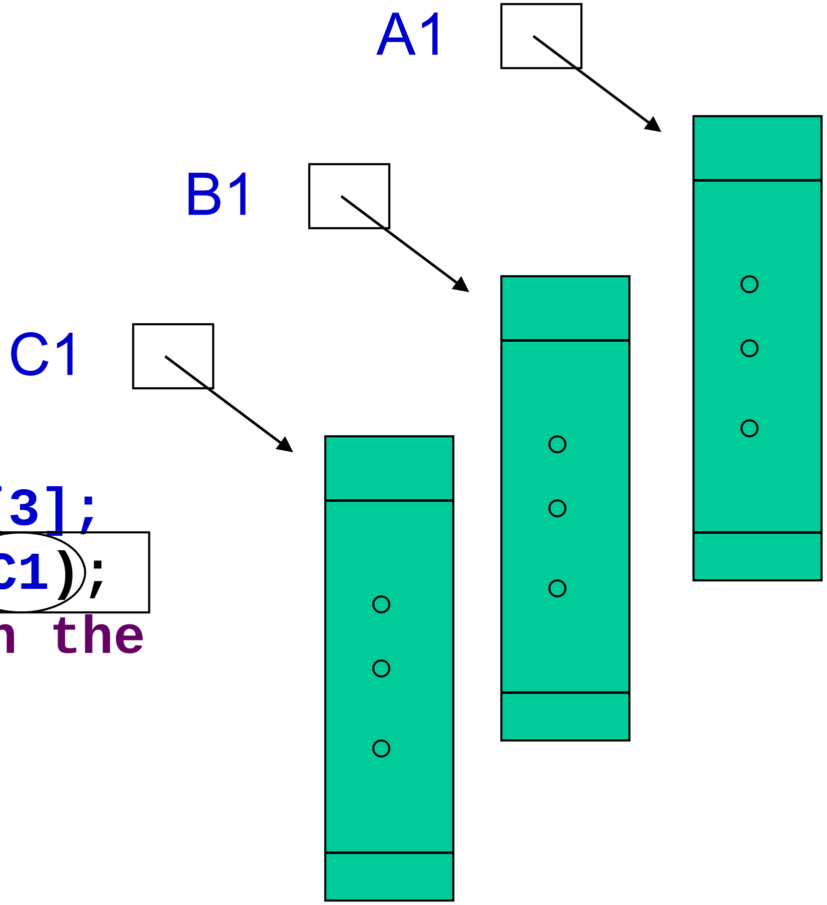


Program Output

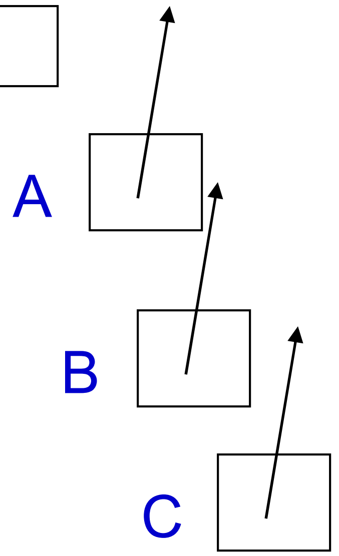
```
Sum of all elements in rows is 210
Sum of all elements in columns is 210
```

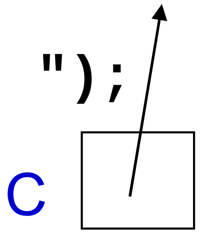
Example 2: Matrix Multiplication

```
public class MatrixMultiArrayApp {
  public static void main(String[] args) {
    int[][] A1 = {
      {1, 2, 3},
      {2, 3, -1},
      {3, -1, 2}};
    int[][] B1 = {
      {1, 2, 3},
      {5, 7, 9},
      {9, 11, 13}};
    int[][] C1 = new int[3][3];
    matrixMultiply(A1, B1, C1);
    // NB: no need to return the
    // resulting matrix C1
    displayMatrix(C1);
  }
```



```
 public static void matrixMultiply(int[][] A,
  int[][] B, int[][] C) {
    int l, m, n;
    for (l = 0; l < A.length; l++)
      for (m = 0; m < B[0].length; m++) {
        C[l][m] = 0;
        for (n = 0; n < A[0].length; n++)
            C[l][m] += A[l][n] * B[n][m];
      }
 }
 public static void displayMatrix(int[][] C) {
   int l, m;
   System.out.println("The product of arrays is: ");
   for (l = 0; l < C.length; l++) {
     for (m = 0; m < C[l].length; m++)
        System.out.print(C[l][m] + " ");
     System.out.println();
   }
 }
}
```





Program Output

```
The product of
arrays is:
38 49 60
8 14 20
16 21 26
```

Chapter 9: Arrays

- Why Learn & Use Arrays?
- Declaring and Creating Arrays
- Operations on Arrays
- Passing Arrays to Methods
- Variable Length Parameter Lists
- Copying Arrays
- Sorting Arrays
- Searching Arrays
- Multidimensional Arrays
- Operations on Multidimensional Arrays
- Passing Multidimensional Arrays to Methods
- **Case Studies**

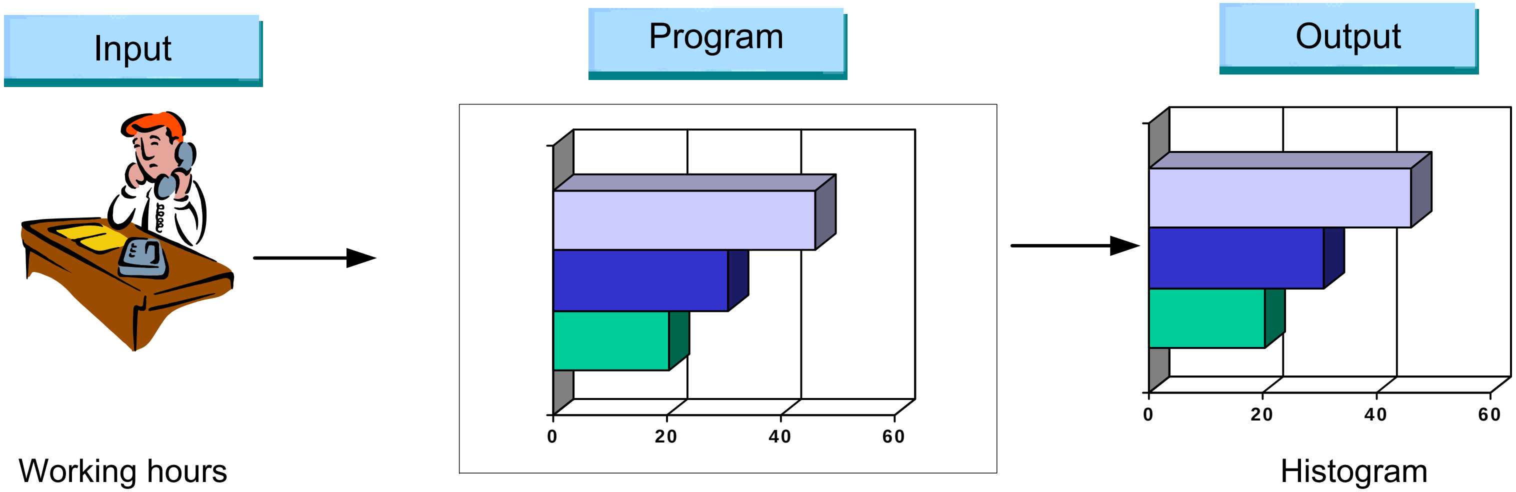
Case Study: Histogram Generator

Problem Specification

Write a program to read employees' working hours, store the input in an array, and then generate a report (a histogram) on the screen.

1-dimensional array

Problem Analysis



Input

Program

Output

Working hours

Histogram

Required inputs:

- The working hours

Required outputs:

- The display of the histogram based on working hours

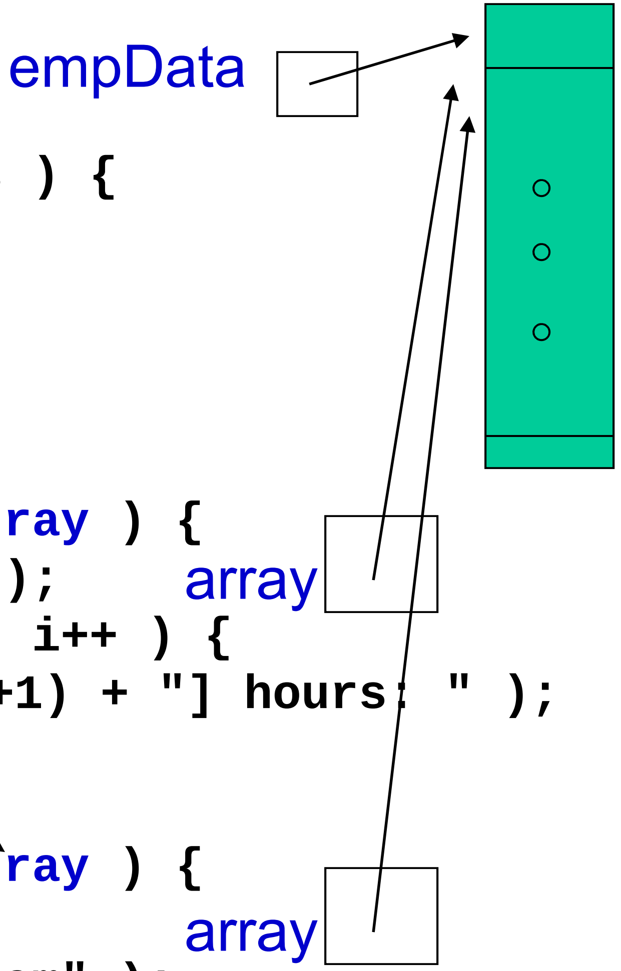
Program Design

Initial Algorithm

1. Read in 10 numbers and store each number into an array.
2. Compute and print out the histogram.

Refer to the textbook for the algorithms in pseudo code.

Implementation



```
import java.util.Scanner;
public class HistogramGenerator {
  public static void main( String[] args ) {
    int[] empData = new int[10] ;
    inputData( empData );
    System.out.println();
    printData( empData );
  }
  public static void inputData( int[] array ) {
    Scanner sc = new Scanner( System.in );
    for ( int i = 0 ; i < array.length ; i++ ) {
      System.out.print( "Worker [" + (i+1) + "] hours: " );
      array[i] = sc.nextInt();
    }  }
  public static void printData( int[] array ) {
    int i,j;
    System.out.print( "Emp No. \tHistogram" );
    for ( i = 0 ; i < array.length ; i++ ) {
      System.out.print( "\t \n  " + (i+1) + "\t\t" );
      for ( j = 0 ; j < array[i] ; j++ )
         System.out.print( '*' ) ;
      System.out.print( "(" + array[i] + ")" );
    }  }
}
```

Testing

Program input and output

```
Worker [1] hours: 5
Worker [2] hours: 6
Worker [3] hours: 4
Worker [4] hours: 7
Worker [5] hours: 8
Worker [6] hours: 9
Worker [7] hours: 1
Worker [8] hours: 1
Worker [9] hours: 2
Worker [10] hours: 3
Emp No.     Histogram
  1   *****(5)
  2   ******(6)
  3   ****(4)
  4   *******(7)
  5   ********(8)
  6   *********(9)
  7   *(1)
  8   *(1)
  9   **(2)
  10  ***(3)
```

Case Study: Computing Production Outputs

Problem Specification

A company employs 20 workers. Each worker works 5 days a week from Monday to Friday. A two-dimensional array of integers, production[20][5], is declared to store the production output for each worker.
For example, production[2][1] indicates the production output for worker with identity 2 on Tuesday.

In this case study, we need to write the following methods:

1. To read the production outputs of all workers for all workdays.

```
    public static void readInput()
```

2. To return the weekly average production output of all the workers.

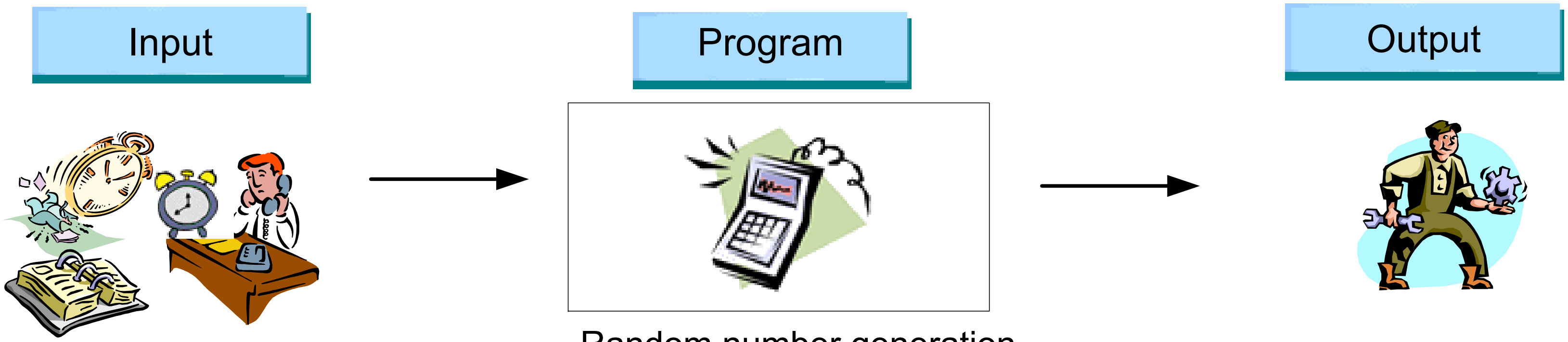
```
    public static double computeAverage()
```

3. To identify the most productive worker for a week.

```
    public static int  findTheBest()
```

Returns index of the best worker.

Problem Analysis



Input

Program

Output

Production outputs

Random number generation
Average = total production outputs/
number of workers

Average production outputs
Most productive worker

Required inputs:

- The production outputs for each day of the week for each worker

Required outputs:

- The average production output
- The most productive worker

Formulas:

- Random num generation function: (int)(Math.random()*100)
- Average output = total output / number of workers

Program Design

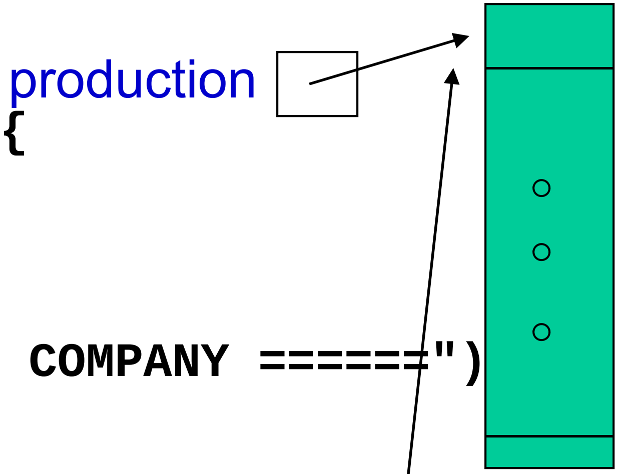
Initial Algorithm

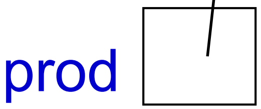
1. Generate the production outputs randomly and store the outputs into a 2-dimensional array production.
2. Print production outputs of workers.
3. Compute the average production output of workers.
4. Print the average production output.
5. Find the best worker.
6. Print the best worker.

Refer to the textbook for the algorithm in pseudocode.

Implementation

```
public class ComputeProduction {
  public static void main(String[] args) {
    int i,j;
    int worker = 0;
    int[][] production = new int[20][5];
    System.out.println("====== PRODUCTION COMPANY ======")
    readInput(production);
    printInput(production);
    System.out.println("Average production: " +
       computeAverage(production));
    worker = findTheBest(production);
    System.out.println("The most prod worker is: " + worker);
  }
  public static void readInput(int[][] prod) {
    int i,j;
    for (i=0; i<20; i++)
      for (j=0; j<5; j++)
        prod[i][j] = (int)(Math.random()*100);
  }
```





Implementation

```
public static void printInput(int[][] prod) {
  int i,j;
  for (i=0; i<20; i++){
    System.out.print("\n");
    for (j=0; j<5; j++){
      System.out.print(prod[i][j] + " ");
    }
  }
  System.out.println();
}
public static double computeAverage(int[][] prod) {
  int i,j;
  int total = 0;
  for (i=0; i<20; i++)
    for (j=0; j<5; j++)
      total += prod[i][j];
  return total/20.0;
}
```

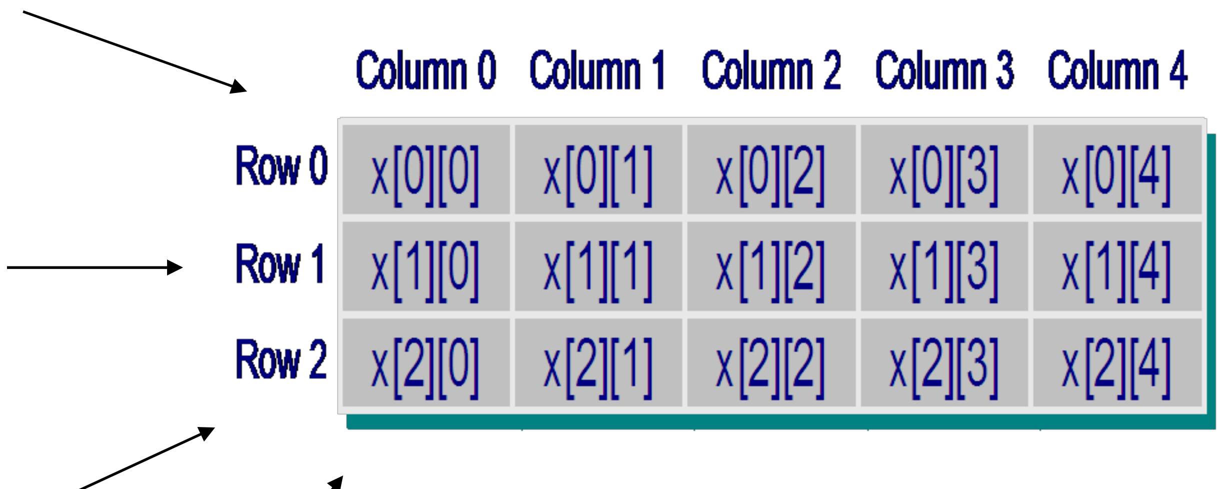
Implementation

```
  public static int findTheBest(int[][] prod) {
    int weekTotal[] = new int[20];
    int best=0;
    int i,j,highest;
    for (i=0; i<20; i++){    // compute total for each worker
      weekTotal[i] = 0;
      for (j=0; j<5; j++)
         weekTotal[i] += prod[i][j];
   }
   highest = -1;
   for ( i=0 ; i<20 ; i++ ) {   // find the highest total
     if ( highest < weekTotal[i] )
          highest = weekTotal[i] ;
   }
   for ( i=0 ; i<20 ; i++ ) {   // identify worker with highest
     if ( highest == weekTotal[i] )
       best = i ;
   }
    return best;
  }
}
```

Any way to speed it up?

Testing: Program input and output

```
====== PRODUCTION COMPANY ======
6 12 15 93 41
4 31 21 58 56
21 28 15 83 1
55 90 56 93 62
22 34  14 50 55
93 2 53 78 48
10 57  9 49 59
28 97  38 52 62
40 0 24 17 45
49 83 35 43 97
65 70  91 63 65
93 62 46 76 90
26 96 85 72 58
41 3 57 11 71
41 55 6 95 33
41 65 14 16 0
9 73 57 24 99
3 22 63 87 39
9 72 69 65 78
19 89 46 28 36
Average production: 238.9
The best production worker is: 11
```



	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]
Row 1	x[1][0]	x[1][1]	x[1][2]	x[1][3]	x[1][4]
Row 2	x[2][0]	x[2][1]	x[2][2]	x[2][3]	x[2][4]

Any problem/assumption in this program?
How if two workers ……

When doing programming,
Think about possible cases!!!

Further Reading

- Read Chapter 9 on “Arrays” of the textbook.
- Read other case studies from the chapter.

Thank you !!!