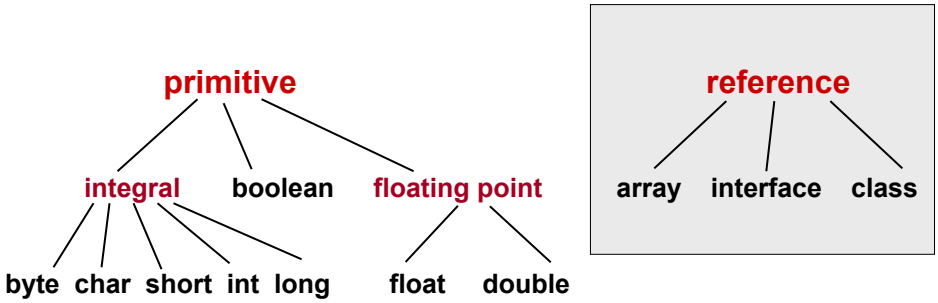
Arrays (vetores) de uma dimensão

Java Data Types

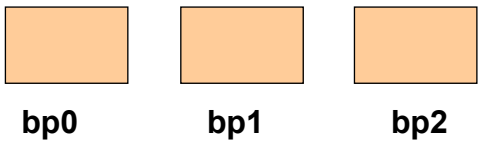


Data Type Categories

- Scalar data type A data type in which
  - **the values are ordered and each value is atomic (indivisible)**
  - **`int`, `float`, `double`, and `char` data types are scalar**
- Ordinal data type A data type in which
  - **each value (except the first) has a unique predecessor**
  - **each value (except the last) has a unique successor**

Three Blood Pressure Readings

```
int  bp0,  bp1,  bp2;
int  total;
```




```
total =  bp0 + bp1 + bp2;
```

Composite Data Type

- Composite data type A data type that **allows a collection of values to be associated with an identifier of that type**
- **There are two forms of composite types: unstructured and structured**
- **In Java, composite types are classes, interfaces, and arrays**

Structured Data Type

- A structured data type is one in which the components are organized with respect to each other
- The organization determines the method used to access individual components
- An array is a structured data type whose components are accessed by position

1000 Blood Pressure Readings

```
int[] bp = new int[1000];
// Declares and instantiates (creates)
// an array of 1000 int values
// and initializes all 1000 elements to zero,
// the default integer value
```

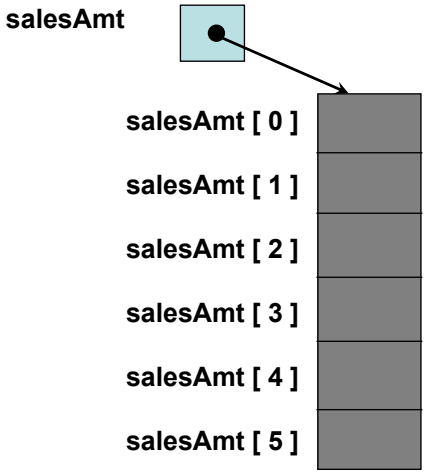
0	0	0	. . . .	0
bp[0]	**bp[1]**	**bp[2]**	**. . . .**	**bp[999]**

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 an array, beginning at the base address
- The base address is stored in the array variable
- 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

```
double[] salesAmt;
salesAmt = new double[6];
```


```
double[] salesAmt;
salesAmt = new double[6];
```



Array Definitions

- Array A collection of homogenous elements, given a single name
- Length A variable associated with the array that contains the number of locations allocated to the array
- Subscript (or index) A variable or constant used to access a position in the array: The first array element always has subscript 0, the second has subscript 1, and the last has subscript length-1
- When allocated, the elements are automatically initialized to the default value of the data type: 0 for primitive numeric types, `false` for `boolean` types, or `null` for references types.

Another 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
```

0.0	0.0	0.0	0.0	0.0
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

indexes or subscripts

Declaring and Allocating an Array

- Operator `new` is used to allocate the specified number of memory locations needed for array DataType

SYNTAX FORMS

DataType[ ] ArrayName; **// declares array**

ArrayName = new DataType [IntExpression]; **// allocates array**

DataType[ ] ArrayName = new DataType [IntExpression];

Assigning values to array elements

```
double[] temps = new double[5];      // Creates array
int m = 4;
temps[2] = 98.6;
temps[3] = 101.2;
temps[0] = 99.4;
temps[m] = temps[3] / 2.0;
temps[1] = temps[3] - 1.2;
 // What value is assigned?
```

99.4	?	98.6	101.2	50.6
temps[0]	**temps[1]**	**temps[2]**	**temps[3]**	**temps[4]**

What values are assigned?

```
double[] temps = new double[5]; // Allocates array
int m;

for (m = 0; m < temps.length; m++)
  temps[m] = 100.0 + m * 0.2;
```

What is `length`*?*

?	?	?	?	?
temps[0]	**temps[1]**	**temps[2]**	**temps[3]**	**temps[4]**

Now what values are printed?

```
final int ARRAY_SIZE = 5;            // Named constant
double[] temps;
temps = new double[ARRAY_SIZE];
int m;
 . . . . .
for (m = temps.length-1; m >= 0; m--)
  System.out.println("temps[" + m + "] = " + temps[m]);
```

100.0	100.2	100.4	100.6	100.8
temps[0]	**temps[1]**	**temps[2]**	**temps[3]**	**temps[4]**

Variable subscripts

```
double[] temps = new double[5];
int m = 3;
 . . . . . .
```

What is `temps[m + 1]` *?*

What is `temps[m] + 1` *?*

100.0	100.2	100.4	100.6	100.8
temps[0]	**temps[1]**	**temps[2]**	**temps[3]**	**temps[4]**

Initializer List

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

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

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

Passing Arrays as Arguments

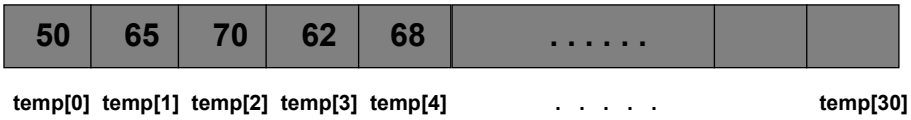
- In Java an array is a reference type. The address of the first item in the array (the base address) is passed to a method with an array parameter
- The name of the array is a reference variable that contains the base address of the array elements
- The array name dot `length` returns the number of locations allocated

Passing an Array as Parameter

```
public static double average(int[] grades)
// Calculates and returns the average grade in an
// array of grades.
// Assumption: All array slots have valid data.
{
  int total = 0;
  for (int i = 0; i < grades.length; i++)
    total = total + grades[i];
  return  (double) total / (double) grades.length;
}
```

Memory allocated for array

```
int[] temps = new int[31];
// Array  holds 31 temperatures
```

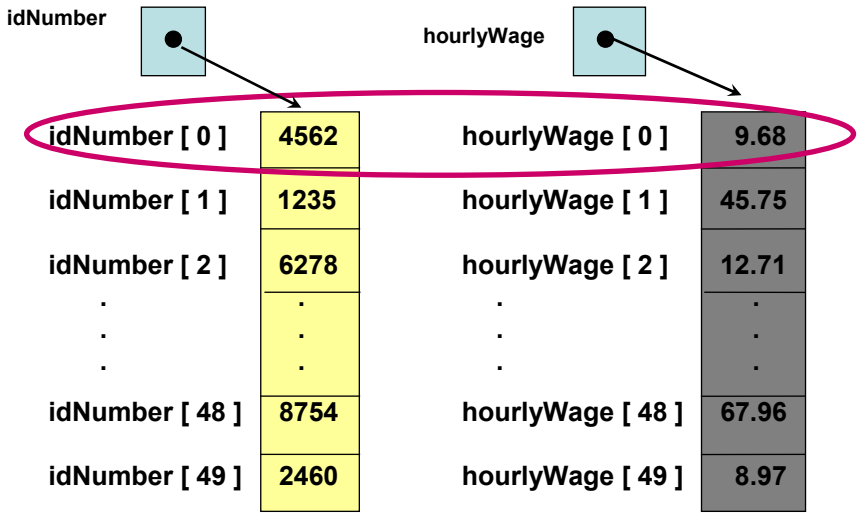


Parallel arrays

- **Parallel arrays Two or more arrays that have the same index range, and whose elements contain related information, possibly of different data types**

```
final int SIZE = 50;
int[] idNumber = new int[SIZE];
float[] hourlyWage = new float[SIZE];
```

```
final  int  SIZE  = 50 ;
int [ ]  idNumber  = new  int  [ SIZE ] ;          // parallel arrays hold
float [ ]  hourlyWage = new  float [ SIZE ] ;    // related information
```



Partial Array Processing

- length is the number of slots assigned to the array
- *What if the array doesn't have valid data in each of these slots?*
- Keep a counter of how many slots have valid data and use this counter when processing the array

Using arrays for counters

- Write a program to count the number of times each letter appears in a text file

letter	ASCII
'A'	65
'B'	66
'C'	67
'D'	68
.	.
.	.
.	.
'Z'	90

datafile.dat

This is my text file.
It contains many things!
5 + 8 is not 14.
Is it?


```
int[ ] letterCount = new int[26];
```

letterCount [0]	2	counts 'A' and 'a'
letterCount [1]	0	counts 'B' and 'b'
.	.	.
.	.	.
.	.	.
letterCount [24]	1	counts ' Y' and 'y'
letterCount [25]	0	counts 'Z' and 'z'

Pseudocode for counting letters

Prepare dataFile
Read one line from dataFile
While not EOF on dataFile
    For each letter in the line
        If letter is am alphabetic character
            Convert uppercase of letter to index
            Increment letterCount[index] by 1
    Read next line from dataFile
Print characters and frequencies to outFile

Frequency Counts

```
String line; line = dataFile.readLine();
// Read one line at a time
int location; char letter;
while (line != null)    // While more data
{ for (location =0; location<line.length(); location++)
  { letter = line.charAt(location);
    if ((letter  >= 'A' && letter <= 'Z') ||
        (letter  >= 'a' && letter <= 'z'))
    { index = (int)Character.toUpperCase(letter) -
            (int) 'A';
      letterCount[index] = letterCount[index] + 1;
    }
  }
  line = dataFile.readLine();      // Get next line
}
```

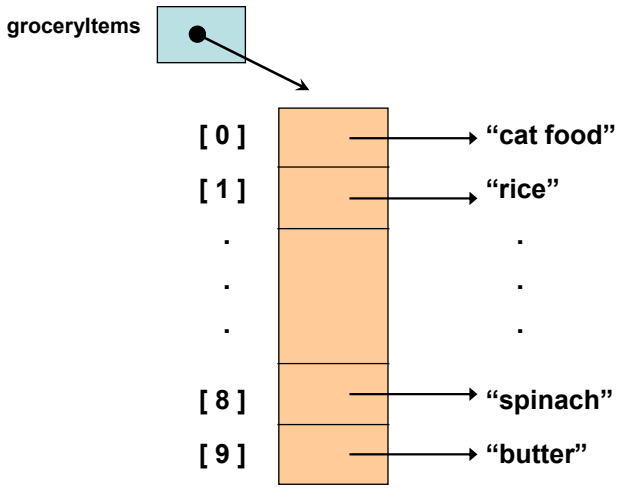
Frequency Counts

```
// print each alphabet letter and its frequency count
for (index = 0; index < letterCount.length; index++)
{
  System.out.println("The total number of "
      + (char) (index + (int) 'A')
      + "'s is ")
      + letterCount[index]);
}
```

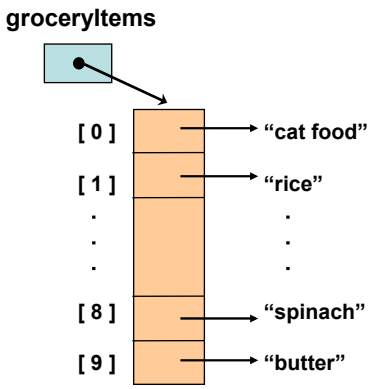

More about Array Indexes

- Array indexes can be any integral expression of type `char`, `short`, `byte`, or `int`
- It is the programmer's responsibility to make sure that an array index does not go out of bounds. The index must be within the range 0 through the array's length minus 1
- Using an index value outside this range throws an `ArrayIndexOutOfBoundsException`; prevent this error by using public instance variable `length`

```
String[] groceryItems = new String[10];
```

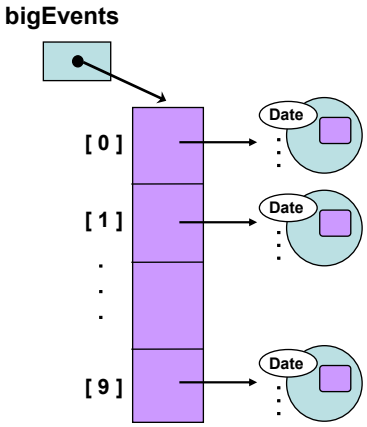



```
String[] groceryItems = new String[10];
```



Expression	Class/Type
groceryItems	Array
groceryItems[0]	String
groceryItems[0].charAt(0)	char

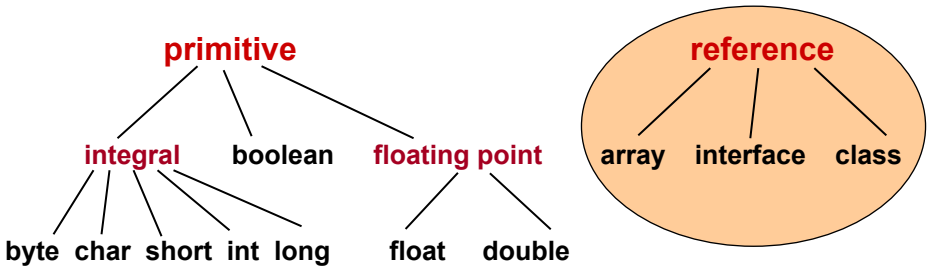
```
Date[] bigEvents = new Date[10];
```



Expression	Class/Type
bigEvents	Array
bigEvents[0]	Date
bigEvents[0].month	String
bigEvents[0].day	int
bigEvents[0].year	int
bigEvents[0].month.charAt(0)	char

Multidimensional Arrays and Numeric Computation

Java Data Types



Two-Dimensional Array

- Two-dimensional array A collection of homogeneous components, structured in two dimensions, (referred to as rows and columns); each component is accessed by a pair of indexes representing the component's position within each dimension

Syntax for Array Declaration

Array Declaration

DataType [ ][ ] ArrayName;

EXAMPLES

```
double[][] alpha;
String[][] beta;
int[][] data;
```

Two-Dimensional Array Instantiation

Two-Dimensional Array Instantiation

ArrayName = new DataType [Expression1] [Expression2];

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

Two-Dimensional Array Instantiation

Two-Dimensional Array Instantiation

ArrayName = new DataType [Expression1] [Expression2];

Two forms for declaration and instantiation

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

OR

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

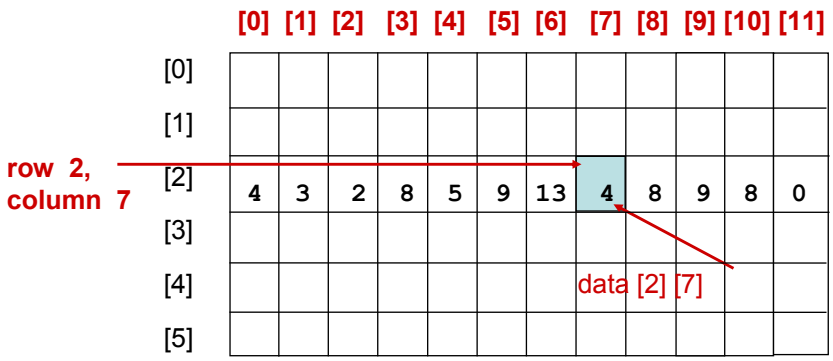
Indexes in Two-Dimensional Arrays

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;
```

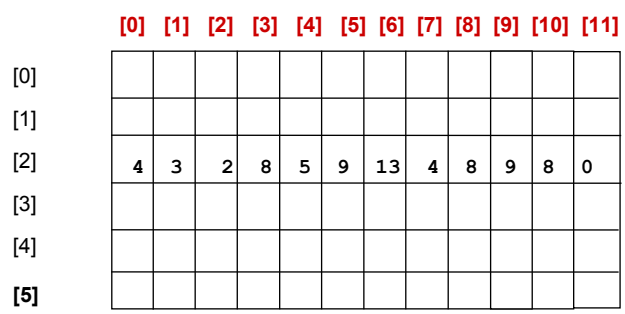


The `length` fields

```
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



Using the `length` field

```
int[][]  data = new  int[6][12];
for (int  i = 0;  i < data[2].length;  i++)
// prints contents of row 2
  System.out.println(data[2][i]);
```

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

EXAMPLE -- Monthly high temperatures

```
final int NUM_STATES = 50;
int[][] stateHighs;
stateHighs = new int [NUM_STATES][12];
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
[0]												
[1]												
[2]	66	64	72	78	85	90	99	105	98	90	88	80
.												
.												
.												
[48]												
[49]												

row 2, col 7 might be Arizona's high for August

stateHighs [2] [7]

Arizona's average high temperature

```
int total = 0;
int month;
int average;
for (month = 0; month < 12; month ++)
  total = total + stateHighs[2][month];
average  = (int) ((double) total / 12.0  + 0.5);
```

average

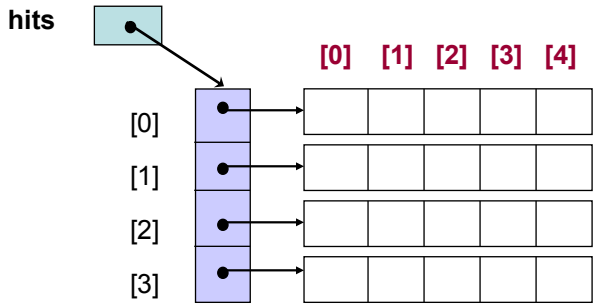
85

Two-Dimensional Array

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

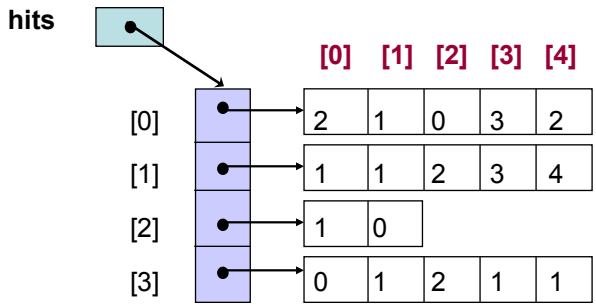
Initializer Lists

```
int[][] hits  =     {{  2,  1,  0,  3,  2 },
                     {  1,  1,  2,  3,  4 },
                     {  1,  0,  0,  0,  0 },
                     {  0,  1,  2,  1,  1 }};
```



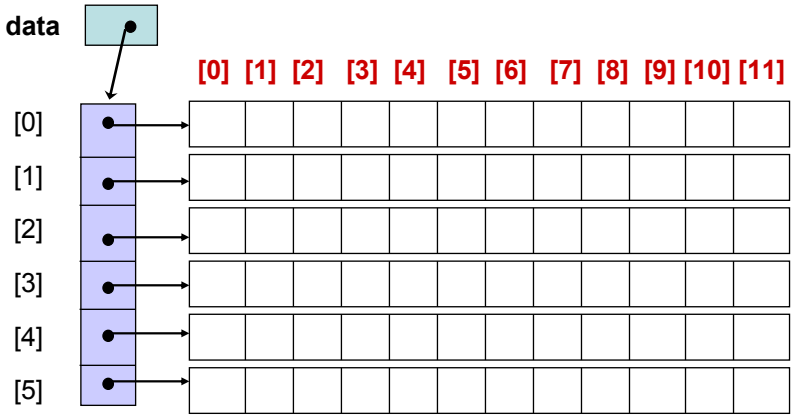
Ragged Arrays

```
int[][] hits  =     {{ 2,  1,  0,  3,  2 },
                     { 1,  1,  2,  3,  4 },
                     { 1,  0 },
                     { 0,  1,  2,  1,  1 }};
```



Java Array Implementation

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

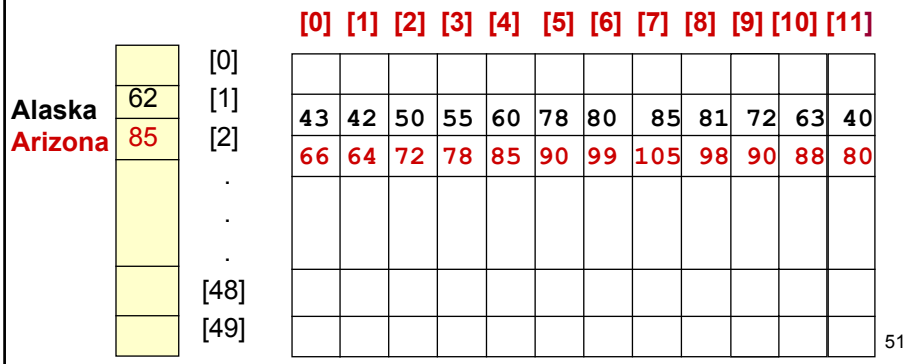


Arrays as parameters

- Just as with a one-dimensional array, when a two- (or higher) dimensional array is passed as an argument, the base address of the argument array is sent to the method
- Because Java has a `length` field associated with each array that contains the number of slots defined for the array, we do not have to pass this information as an additional parameter

stateHighs and stateAverages

```
final int NUM_STATES  =  50;
int[][] stateHighs = new int[NUM_STATES][12];
int  stateAverages [NUM_STATES];
```



Code to calculate averages for each state

```
public static void  findAverages(int[][] stateHighs,
   int[] stateAverages)
// Result: stateAverages[0..NUM_STATES] contains
// rounded average high temperature for each state
{ int state, month, total;
  for (state = 0; state < stateAverages.length;state++)
  { total = 0;
    for (month = 0; month < 12; month++)
      total = total + stateHighs[state][month];
    stateAverages[state] = (int)((double)total/12.0+0.5);
   }
}
```

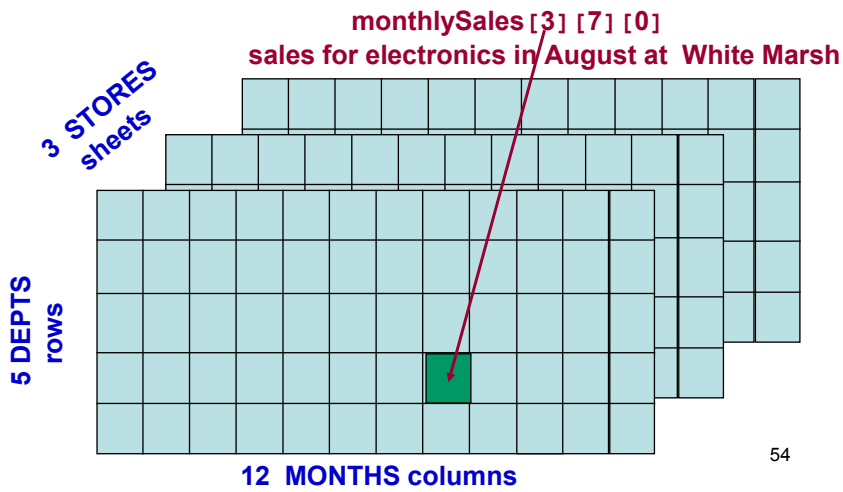
Declaring Multidimensional Arrays

EXAMPLE OF THREE-DIMENSIONAL ARRAY

```
final int  NUM_DEPTS  =  5;
// mens, womens, childrens, electronics, furniture

final int  NUM_STORES   =    3;
// White Marsh,  Owings Mills, Towson

int[][][]  monthlySales;
monthlySales =  new int[NUM_DEPTS][12][NUM_STORES];
```

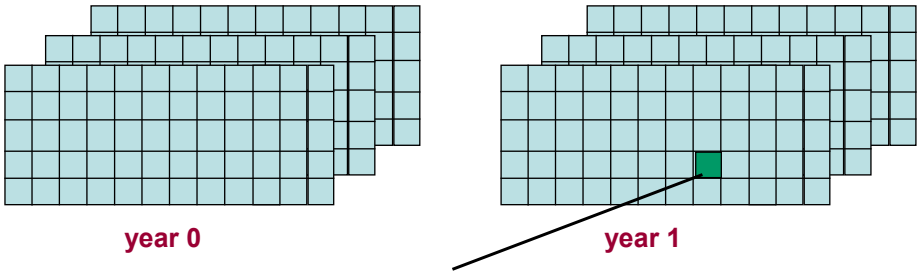
rows columns sheets


```
final int NUM_DEPTS = 5;
final int NUM_STORES = 3;
int[][][]   monthlySales;
monthlySales = new int[NUM_DEPTS][12][NUM_STORES];
```



Adding a fourth dimension . . .

```
final int NUM_DEPTS = 5;
final int NUM_STORES = 3;
final int NUM_YEARS = 2;
int[][][][] moreSales;
moreSales =  new  int[NUM_DEPTS][12][NUM_STORES][NUM_YEARS];
```



moreSales[3] [7] [0] [1]

for electronics, August, White Marsh, one year after starting year

Vector Class

- Vector Class A built-in class in `java.util` that offers functionality similar to that of a one-dimensional array with the general operations similar to those we have provided for our list classes

Exemplo

```
// Nome do pacote
package rrio.class_loader;


// bibliotecas necessarias
import java.util.*;
import java.io.*;
import java.lang.Runtime.*;
import lti.java.jcf.*;
import lti.java.javadump.*;

/**
 * O objetivo desta classe e a implementacao de um class loader capaz
 * de carregar classes de varios locais, como arquivos locais ou URLs.
 *
 * Esta classe e derivada da originalmente escrita por Jack Harich,
 * que pode ser encontrada em
 * http://www.javaworld.com/javaworld/javatips/jw-javatip39.html
```

Exemplo

```
...

/**
  * Hashtable que funciona como cache para classes ja lidas. Note
  * que so teremos uma hashtable para toda a classe.
  */
 private Hashtable classes = new Hashtable();

...

// Define o array de bytes lidos, transformando-o em uma classe
    result = defineClass(newName, newClassBytes, 0, newClassBytes.length);
    if (result == null) {
       throw new ClassFormatError();
    }

    // Classe lida. Coloca-se agora o resultado na cache.
    classes.put(newName, result);
```

Exemplo

```
...

// Verifica na cache local de classes a existencia da classe className
    result = (Class) classes.get(className);
    if (result != null) {
       return result;
    }

...

public Enumeration getLoadedClasses () {
    return classes.keys();
  }
```

Exemplo

```
...

 /**
   * Vetor que armazena a localizacao dos configuradores locais.
   */
  private static Vector configuradores_locais = new Vector();



 // verifica se o host ja' foi incluido no vetor de configuradores locais.
      // no caso de nao ter sido incluido, realiza sua inclusao
      if (!configuradores_locais.contains(host))
         configuradores_locais.addElement(host);
    }
```

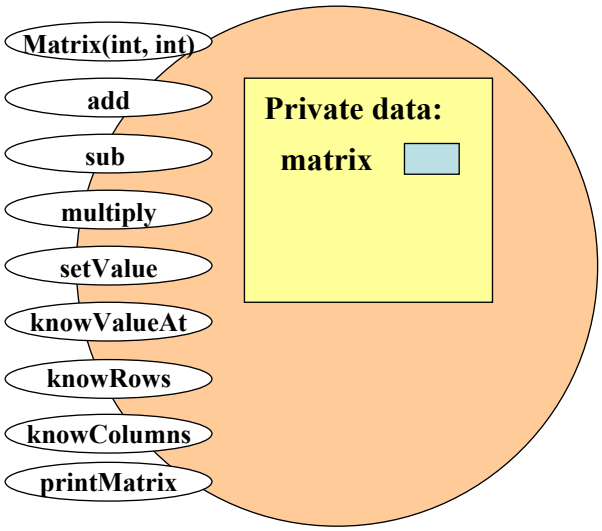
Exemplo

```
...

 // Se origem igual a URL
  if (origem.equals("URL")) {
   for (int i = 0; i< configuradores_locais.size(); i++) {

    try {
     // Se conecta ao configurador local da maquina host1
     configurador_local myServer = (configurador_local) Naming.lookup
                                   ("rmi://" +
                                    configuradores_locais.elementAt(i)
                                    + "/" + "Configurador Local");
     // realiza chamada remota
     result = myServer.setURL(base);
    }
    catch (Exception ex) {
     return "Erro: " + ex;
    }
   }
```

Floating-Point Numbers

- Precision The maximum number of significant digits that can be represented in the form used
- Significant digits Those digits from the first nonzero digit on the left to the last nonzero digit on the right plus any 0 digits that are exact
- Representational error An arithmetic error that occurs when the precision of the true result of an arithmetic operation is greater than the precision of the machine

Matrix Class



Finding a matrix product

this.matrix

$$\begin{bmatrix} 1 & 2 & 0 & 0 \\ 2 & 0 & 4 & 0 \\ 0 & 5 & 0 & 6 \end{bmatrix}$$

two.matrix

$$\begin{bmatrix} 0 & 2 \\ 4 & 0 \\ 0 & 3 \\ 0 & 1 \end{bmatrix}$$

=

result.matrix

$$\begin{bmatrix} 8 & 2 \\ 0 & 16 \\ 20 & 6 \end{bmatrix}$$

this.matrix[0].length == two.matrix.length

```
Matrix  result = new Matrix (this.matrix.length, two.matrix[0].length);
```

Class MatException

```
//  Defines an Exception class for Matrix errors
package  matrix;

public class MatException extends Exception
{
  public MatException()
  {
    super();
  }
  public MatException(String message)
  {
    super(message);
  }
}
```

Using MatException

```
// Wrong sizes for addition
if (matrix.length != two.matrix.length ||
      matrix[0].length != two.matrix[0].length)
        throw new MatException("Illegal addition.");
...
// Addition overflow
  result.matrix[row][col] = matrix[row][col]
                            + two.matrix[row][col];
  if (Double.isInfinite(result.matrix[row][col]))
     throw new MatException("Addition overflow");
```