Arrays (vetores) de uma dimensão

Java Data Types

primitive
integral
byte char short int long

boolean

floating point
float double

reference
array interface class
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

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

bp0 bp1 bp2

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

```java
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
```

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td>...</td>
<td>0</td>
</tr>
</tbody>
</table>


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];
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.

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

<table>
<thead>
<tr>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
<th>0.0</th>
</tr>
</thead>
</table>


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
```

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;
// What value is assigned?
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>99.4</td>
<td>?</td>
<td>98.6</td>
<td>101.2</td>
<td>50.6</td>
</tr>
</tbody>
</table>
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?


Now what values are printed?

final int ARRAY_SIZE = 5; // Named constant
double[] temps;
tempre = 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
Variable subscripts

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

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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>100.0</td>
<td>100.2</td>
<td>100.4</td>
<td>100.6</td>
<td>100.8</td>
</tr>
</tbody>
</table>

Initializer List

```java
int[] ages = {40, 13, 20, 19, 36};
for (int i = 0; i < ages.length; i++)
    System.out.println("ages[" + i + "] = " + ages[i]);
```

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>13</td>
<td>20</td>
<td>19</td>
<td>36</td>
</tr>
</tbody>
</table>
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

```java
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

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

<table>
<thead>
<tr>
<th>50</th>
<th>65</th>
<th>70</th>
<th>62</th>
<th>68</th>
<th>......</th>
<th>......</th>
<th>temp[30]</th>
</tr>
</thead>
</table>

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

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

<table>
<thead>
<tr>
<th>idNumber [0]</th>
<th>4562</th>
<th>hourlyWage [0]</th>
<th>9.68</th>
</tr>
</thead>
<tbody>
<tr>
<td>idNumber [1]</td>
<td>1235</td>
<td>hourlyWage [1]</td>
<td>45.75</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>idNumber [48]</td>
<td>8754</td>
<td>hourlyWage [48]</td>
<td>67.96</td>
</tr>
<tr>
<td>idNumber [49]</td>
<td>2460</td>
<td>hourlyWage [49]</td>
<td>8.97</td>
</tr>
</tbody>
</table>

### 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

**ASCII**

<table>
<thead>
<tr>
<th>letter</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>‘A’</td>
<td>65</td>
</tr>
<tr>
<td>‘B’</td>
<td>66</td>
</tr>
<tr>
<td>‘C’</td>
<td>67</td>
</tr>
<tr>
<td>‘D’</td>
<td>68</td>
</tr>
<tr>
<td>‘E’</td>
<td>69</td>
</tr>
<tr>
<td>‘F’</td>
<td>70</td>
</tr>
<tr>
<td>‘G’</td>
<td>71</td>
</tr>
<tr>
<td>‘H’</td>
<td>72</td>
</tr>
<tr>
<td>‘I’</td>
<td>73</td>
</tr>
<tr>
<td>‘J’</td>
<td>74</td>
</tr>
<tr>
<td>‘K’</td>
<td>75</td>
</tr>
<tr>
<td>‘L’</td>
<td>76</td>
</tr>
<tr>
<td>‘M’</td>
<td>77</td>
</tr>
<tr>
<td>‘N’</td>
<td>78</td>
</tr>
<tr>
<td>‘O’</td>
<td>79</td>
</tr>
<tr>
<td>‘P’</td>
<td>80</td>
</tr>
<tr>
<td>‘Q’</td>
<td>81</td>
</tr>
<tr>
<td>‘R’</td>
<td>82</td>
</tr>
<tr>
<td>‘S’</td>
<td>83</td>
</tr>
<tr>
<td>‘T’</td>
<td>84</td>
</tr>
<tr>
<td>‘U’</td>
<td>85</td>
</tr>
<tr>
<td>‘V’</td>
<td>86</td>
</tr>
<tr>
<td>‘W’</td>
<td>87</td>
</tr>
<tr>
<td>‘X’</td>
<td>88</td>
</tr>
<tr>
<td>‘Y’</td>
<td>89</td>
</tr>
<tr>
<td>‘Z’</td>
<td>90</td>
</tr>
</tbody>
</table>

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

datafile.dat

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

- letterCount[0] counts ‘A’ and ‘a’
- letterCount[1] counts ‘B’ and ‘b’
- letterCount[24] counts ‘Y’ and ‘y’
- letterCount[25] 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 an 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

```java
// 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];

groceryItems

[0] "cat food"
[1] "rice"
.. .. .. ..
[8] "spinach"
[9] "butter"

<table>
<thead>
<tr>
<th>Expression</th>
<th>Class/Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>groceryItems</td>
<td>Array</td>
</tr>
<tr>
<td>groceryItems[0]</td>
<td>String</td>
</tr>
<tr>
<td>groceryItems[0].charAt(0)</td>
<td>char</td>
</tr>
</tbody>
</table>
Date[] bigEvents = new Date[10];

<table>
<thead>
<tr>
<th>Expression</th>
<th>Class/Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigEvents</td>
<td>Array</td>
</tr>
<tr>
<td>bigEvents[0]</td>
<td>Date</td>
</tr>
<tr>
<td>bigEvents[0].month</td>
<td>String</td>
</tr>
<tr>
<td>bigEvents[0].day</td>
<td>int</td>
</tr>
<tr>
<td>bigEvents[0].year</td>
<td>int</td>
</tr>
<tr>
<td>bigEvents[0].month.charAt(0)</td>
<td>char</td>
</tr>
</tbody>
</table>

Multidimensional Arrays and Numeric Computation
Java Data Types

- **primitive**
  - integral
    - byte
    - char
    - short
    - int
    - long
  - boolean
  - floating point
    - float
    - double

- **reference**
  - array
  - interface
  - class

---

**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 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

```java
int[][] data;
data = new int[6][12];
data[2][7] = 4;
```

The length fields

```java
int[][] data = new int[6][12];
data.length  // gives the number of rows in array data
data[2].length  // gives the number of columns in row 2
```
Using the length field

```java
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]);
```

![Diagram of a 6x12 array with row 2 highlighted]

EXAMPLE -- Monthly high temperatures

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

![Diagram of a 50x12 array with row 2, column 7 highlighted]
Arizona’s average high temperature

```java
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);
```

Two-Dimensional Array

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

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

Ragged Arrays

```c
int[][] hits = {{ 2, 1, 0, 3, 2 },
                { 1, 1, 2, 3, 4 },
                { 1, 0 },
                { 0, 1, 2, 1, 1 }};
```
Java Array Implementation

```java
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

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>43</td>
<td>42</td>
<td>50</td>
<td>55</td>
<td>60</td>
<td>78</td>
<td>80</td>
<td>85</td>
<td>81</td>
<td>72</td>
<td>63</td>
<td>40</td>
</tr>
<tr>
<td>1</td>
<td>66</td>
<td>64</td>
<td>72</td>
<td>78</td>
<td>85</td>
<td>90</td>
<td>99</td>
<td>105</td>
<td>98</td>
<td>90</td>
<td>88</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Alaska
Arizona

Code to calculate averages for each state

```java
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**

```java
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**: 5
- **columns**: 12
- **sheets**: 3

sales for electronics in August at **White Marsh**
Adding a fourth dimension ...

```java
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];
```

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
 */

private Hashtable classes = new Hashtable();

...
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 máquina 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;
        }
    }
}
Matrix Class

Matrix(int, int)

- add
- sub
- multiply
- setValue
- knowValueAt
- knowRows
- knowColumns
- printMatrix

Private data:
- matrix

Finding a matrix product

```
this.matrix = [[1 2 0 0],
               [2 0 4 0],
               [0 5 0 6]]

two.matrix = [[0 2],
               [4 0],
               [0 3],
               [0 1]]

result.matrix = [8 2]
                 [0 16]
                 [20 6]
```

```
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");