School of Computer Science

UNIVERSIDAD DE OVIEDO

Lesson 6: Grouping objects. Fixed-size collections

Introduction to Programming

Academic year 2023-2024

Concepts

- **Fixed-size collections**
- **The for loop**
- **Iterators**

Fixed-size collections

- Sometimes, we can predefine the maximum size for a collection.
- Programming languages provide such collections, known as **arrays**.
- Arrays use a **special syntax**.
- Some advantages:
  - Accessing **items from an array** is usually more efficient than accessing items in a flexible size collection.
  - **Arrays** can store either references to objects or primitive data type values. Flexible size collections can only store references to objects.

Analyzing a web log

- Web servers maintain log files storing information about the user accesses to the website.
- Processing them a webmaster can:
  - Find the most popular web pages.
  - Find the most active moments in a day, week or month.
  - Know the data amount exchanged with the clients.
  - Find broken links.

Analyzing a web log

```
81.9.205.73 - - [01/Oct/2020:17:03:05 +0100] "GET /~falvarez HTTP/1.1" 301 368 "-"
"Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5"

81.9.205.73 - - [01/Oct/2020:17:03:05 +0100] "GET /~falvarez/ HTTP/1.1" 200 2358 "-"
"Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US; rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5"

81.9.205.73 - - [01/Oct/2020:17:03:05 +0100] "GET /icons/blank.gif HTTP/1.1" 200 148
"http://156.35.98.175/~falvarez/" "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5"

81.9.205.73 - - [01/Oct/2020:17:03:05 +0100] "GET /icons/back.gif HTTP/1.1" 200 216
"http://156.35.98.175/~falvarez/" "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5"

81.9.205.73 - - [01/Oct/2020:17:03:05 +0100] "GET /icons/folder.gif HTTP/1.1" 200 225
"http://156.35.98.175/~falvarez/" "Mozilla/5.0 (Windows; U; Windows NT 6.0; en-US;
rv:1.9.0.5) Gecko/2008120122 Firefox/3.0.5"
```

Creating an array object

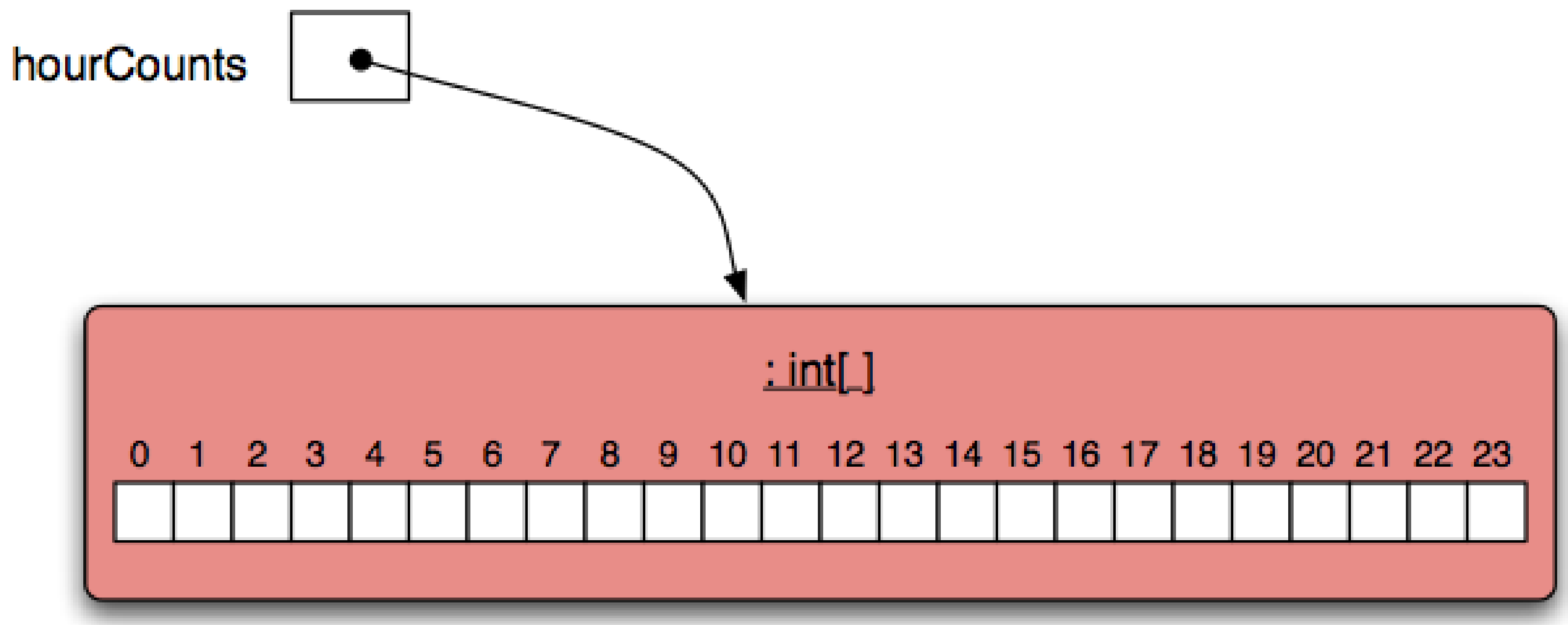
- Example of a simple weblog analyzer.
- Each line in the file contains both the date and the time for an access to the website.

```
public class LogAnalyzer {
    private int[] hourCounts;
     // Stores the number of accesses in each hour

    private LogfileReader reader;

    public LogAnalyzer()
        hourCounts = new int[24];
        reader = new LogfileReader();
    }
    ...
}
```

Declaration for an array variable

Creating the array object

Creating an array object



This is an array of integer numbers

Creating an array object

```
new type[integer-expression]
```

The type for the items to be stored in the array

Array size (can be 0)

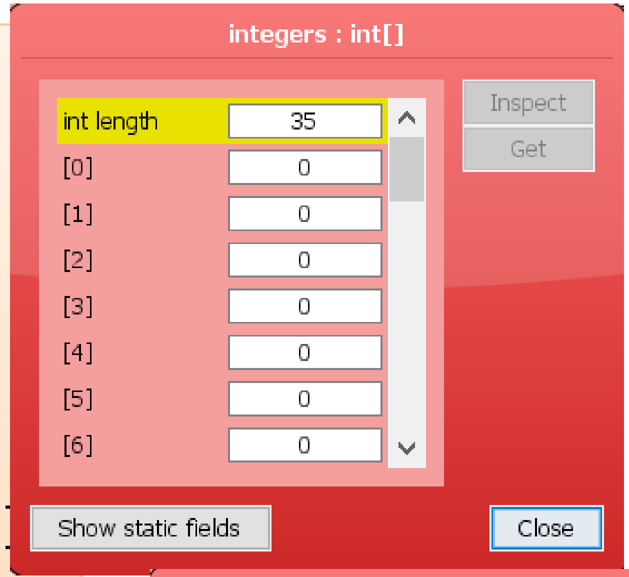
- **Each item in the array is initialized to its default value**

Examples

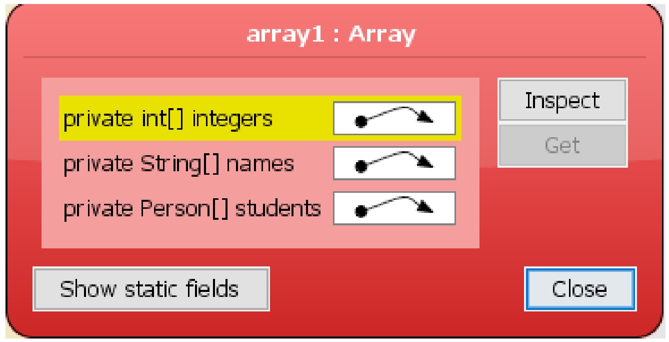
```
String [ ] names;
names = new String[10];

String[] emptyArray = new String[0];
```

Default array initialization

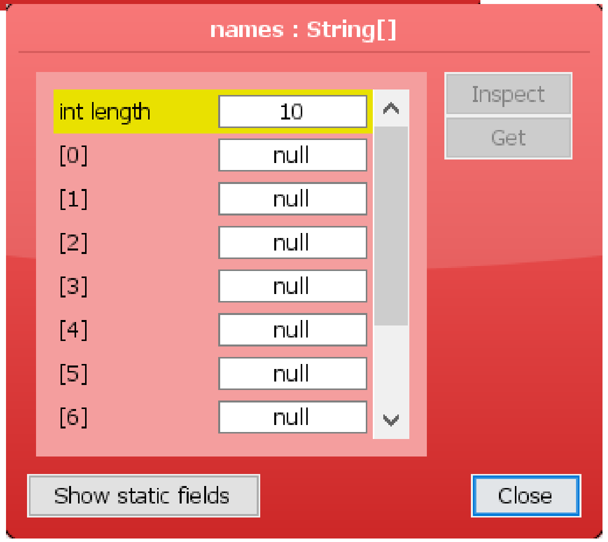
```
public class Array {
    private int[] integers;
    private String[] names;
    private Person[] students;

    public Array() {
      integers = new int[35];
      names = new String[10];
      students = new Person[25]
    }
    ...
}
```



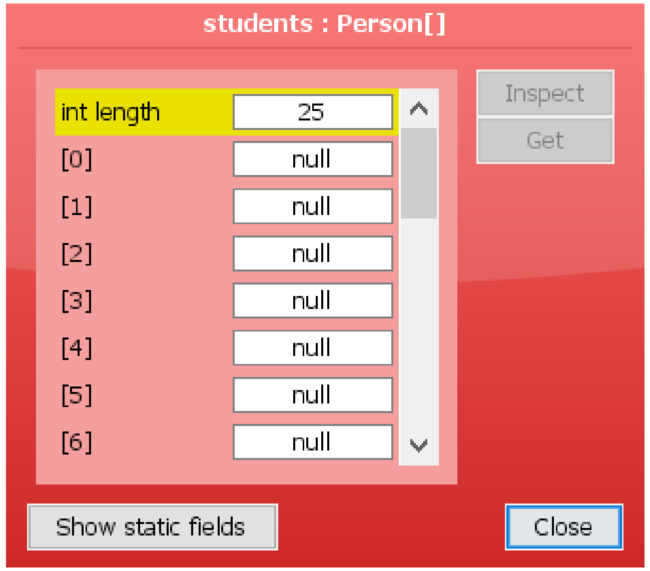
int length	35
[0]	0
[1]	0
[2]	0
[3]	0
[4]	0
[5]	0
[6]	0



private int[] integers
private String[] names
private Person[] students



int length	10
[0]	null
[1]	null
[2]	null
[3]	null
[4]	null
[5]	null
[6]	null



int length	25
[0]	null
[1]	null
[2]	null
[3]	null
[4]	null
[5]	null
[6]	null

Exercises

- Write the declaration for an array variable named **`people`** which could be used to reference one array storing **`Person`** objects.
- Write the declaration for an array variable named **`vacancies`** to store a reference to an array of boolean values.

Exercises

- What’s the meaning of the following declarations?

```
double[] readings;
String[] urls;
```

- How many String objects are created with the following declaration?

```
String [] labels = new String[20];
```

- Find and correct the error in the following declaration:

```
double [] prices= new double(50);
```

Using an array

- Square brackets are used to access items in an array:

  `hourCounts[...]`

- We need an **index** to access the items
- Items in an array can be used as ordinary variables.
  - In the left part of an assignment sentence:

    ```
    hourCounts[hour] = ...;
    ```

  - In an expression:

    ```
    adjust = hourCounts[hour] – 3;
    hourCounts[hour]++;
    ```

Common use for an array

```
private int[] hourCounts;
private String[] names;
```

Declaration

```
...

hourCounts = new int[24];
```

Creation

```
...

hourCounts[i] = 0;
hourCounts[i]++;
System.out.println(hourCounts[i]);
```

Use

Literal arrays

```
private int[] numbers = { 4, 8, 15, 16, 23, 42};
```

Declaration and initialization

```
System.out.println(numbers[i]);
```

```
pos = Arrays.binarySearch(new int[]{1, 3, 5}, 5);
```

Anonymous array

Length of an array

```
private int[] numbers = { 4, 8, 15, 16, 23, 42};

int n = numbers.length;
```

No brackets

- **`length`** is not a method but an attribute. Every array has it and contains its length (the maximun amount of elements it can hold).

The `for` loop

- The are two “flavors”: **for-each** (previous Unit) and **for**.
- The second variant is used when:
  - We need to repeat a sentence or block an exact number of times.
  - We need a variable to be increased (or decreased) by a fixed amount each time.

The `for` loop in pseudo code

Common syntax for the `for` loop

```
for (initialization; condition; modifying action) {
    sentences to be repeated in each iteration
}
// initialization, condition and modifying action are all optional
```

Equivalent code with a `while` loop

```
initialization;
while (condition) {
   sentences to be repeated in each iteration
   modifying action
}
```

An example in Java

Version with `for`

```
for (int hour = 0; hour < hourCounts.length; hour++) {
    System.out.println(hour + ": " + hourCounts[hour]);
}
```

Version with `while`

```
int hour = 0;
while (hour < hourCounts.length) {
    System.out.println(hour + ": " + hourCounts[hour]);
    hour++;
}
```

Version with `for-each`

```
for(int value : hourCounts) {// does not print ‘hour’
    System.out.println(": " + value);
}
```

More examples

```
// infinite loop
for (;;) { … }

// multiple initialization statements and modifying actions
int x;
int y;
for (x = 3, y = 4; x + y < 15; x++, y++) { … }

// no initialization nor modifying actions
Random random=new Random();
for (; random.nextInt(10) < 5; )
   System.out.println("new iteration");

// nested loops
for (int i=1; i <= 10; i++)
   for (int j=1; j <= 10; j++)
         System.out.println(i + " times " + j + " is " + i*j);
```

Exercises

- What is this loop doing?

```
for (int num = 3; num < 40; num = num + 3) {
    System.out.println(num);
}
```

Exercises

- Given an array containing numbers, show all them using a **`for`** and a **`foreach`** loop.

```
int[] numbers = { 4, 8, 15, 16, 23, 42};

for (…;…;…) ...

for (item : collection) ...
```

Review

- **Arrays** are useful when we need a fixed size collection.
- They have a special syntax.
- **`for`** loops are a useful alternative to while loops when we know the number of iterations.
- **`for`** loops are used when we need an index variable.

Review

- **for-each loop**
  - Used to process the whole collection.
  - Can be used with flexible-size and fixed-size collections.
- **for (…;…;…)** loop
  - Can be used to process totally or partially a collection.
  - Can be used with flexible-size and fixed-size collections.
  - Can be used just to repeat the execution of a block of sentences without considering collections.

Review

- **while loop**
  - Can be used to process totally or partially a collection.
  - Can be used with flexible-size and fixed-size collections.
  - Can be used just to repeat the execution of a block of sentences without considering collections.
- **iterator** object
  - Can be used to process totally or partially a collection.
  - Implemented by all Java Class Library collection classes.
  - Commonly used with collections where indexed access is inefficient or impossible.

Two-dimensional arrays (Matrices)

***One dimension*:**

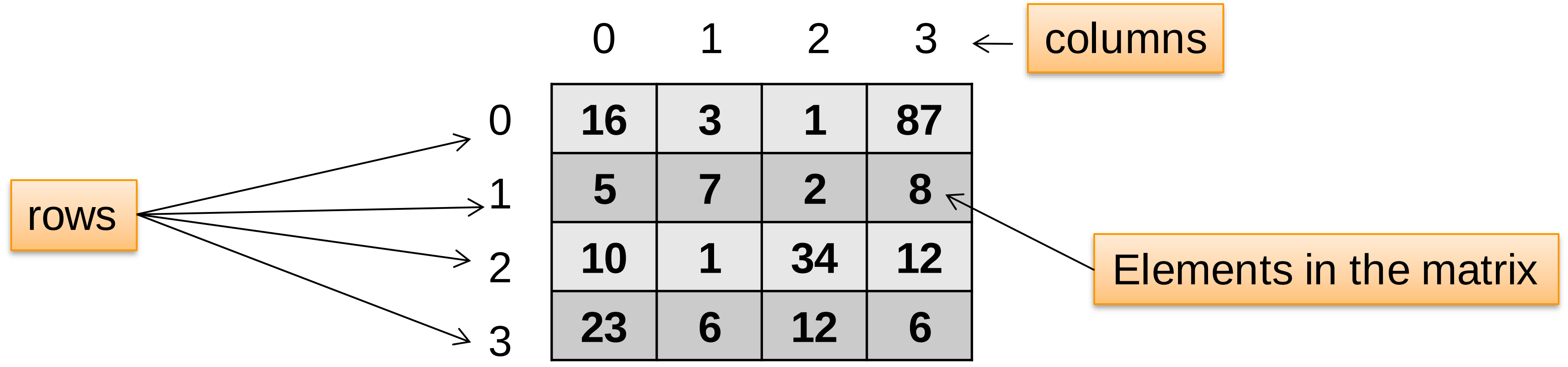
```
type[] myArray = new type[size];

type[] myArray = {e1, e2, e3, …};
```

***Two dimensions*:**

```
type[][] myArray =new type[size1][size2];

type[][] myArray ={{e1, e2, …}, …};
```

- In Java, a **matrix** is an **array of arrays** of the basic type or, better, an **array of references to arrays (rows)** of the basic type.

Two-dimensional arrays (Matrices)



	0	1	2	3
0	16	3	1	87
1	5	7	2	8
2	10	1	34	12
3	23	6	12	6

```
int[][] matrix;
```

Matrix declaration

```
matrix = new int[4][4];
```

Matrix creation

```
matrix[0][0]   is 16
matrix [0][1]  is 3
…
```

Accessing a matrix position

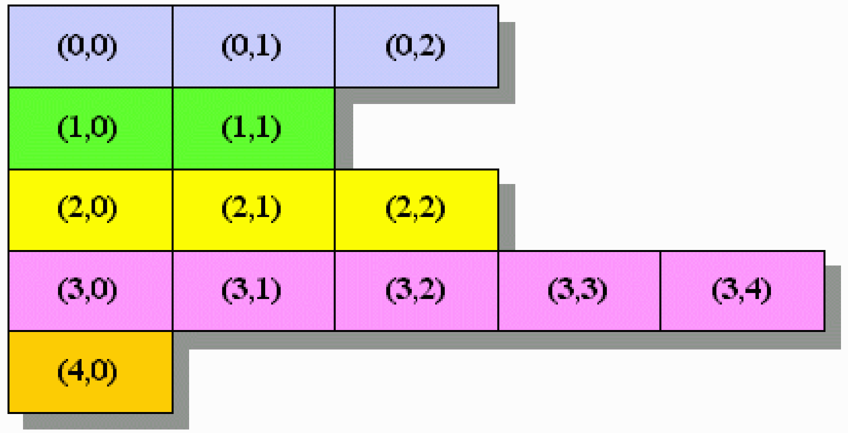
```
matrix[3][3] = 6;
```

Assigning a value to a matrix position

Two-dimensional arrays (Matrices)

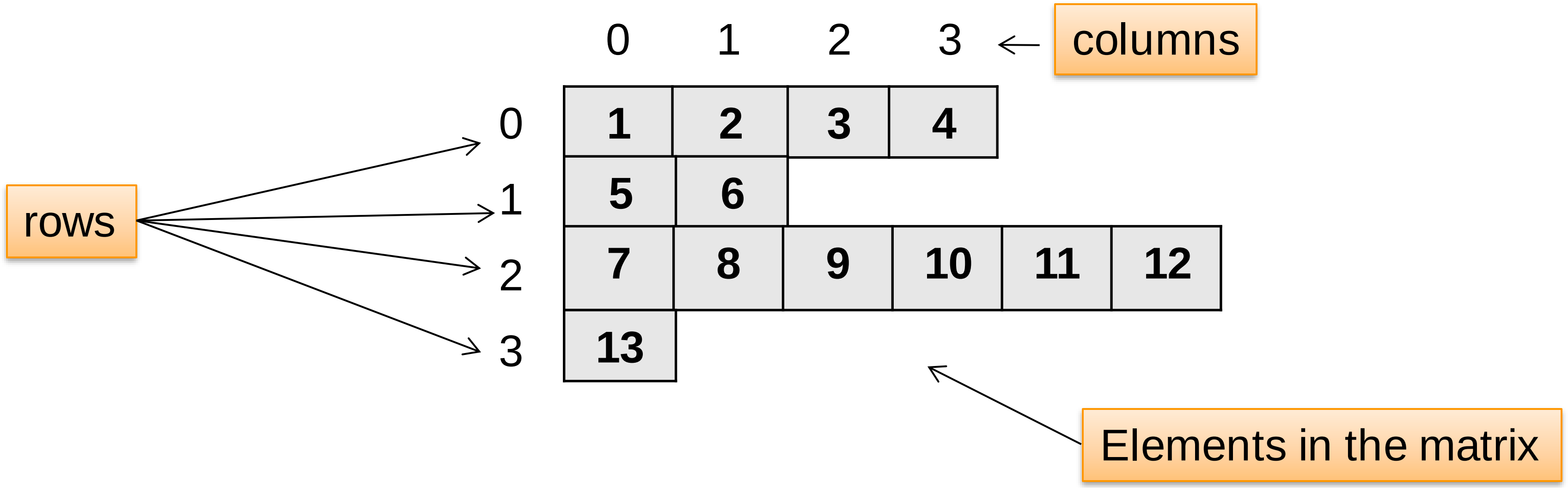
- Unlike other languages, rows/columns can be of different sizes in Java

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

x [0] = new int [3];
x [1] = new int [2];
x [2] = new int [3];
x [3] = new int [5];
x [4] = new int [1];
```



5 “rows” with different size

Two-dimensional arrays (Matrices)



```
int [][] matrix={{1,2,3,4},{5,6},{7,8,9,10,11,12},{13}};
```

Declaration, creation and initialization

Accessing the elements in an array

Filling an array

```
for (int i=0; i < matrix.length; i++) {
   for (int j=0; j < matrix[i].length; j++) {
      matrix[i][j] = produceRandomNumber();
    }
}
```

Number of "rows"

Number of elements in each "row"

Method that returns an integer number

Showing the elements

```
for (int i=0; i < matrix.length; i++) {
   for (int j=0; j < matrix[i].length; j++){
      System.out.print(matrix[i][j]+"\t");
   }
   System.out.println("");
}
```

Review

- Arrays are useful when we need a fixed size collection.
- They have a special syntax.
- **`for`** loops are a useful alternative to while loops when we know the number of iterations.
- **`for`** loops are used when we need an index variable.