Programació I - Tema 6 - Taules/Arrays

UNIVERSITAT DE BARCELONA

Grau en Enginyeria Informàtica
Facultat de Matemàtiques i Informàtica
Curs 18-19

6.1 Introducció

- Hi ha problemes que requereixen tipus de dades més complexes que els existents com a tipus bàsics valor o primitius [1] (*int, char, float...*), per tant es necessiten utilitzar tipus referència: classes ja existents (*String*), **nous tipus de dades** (creant noves **classes** d'**objectes**) i tipus **bàsics NO primitius (*array*)**.
- En aquest tema veurem el tipus referència *array* (concepte de taula).

[1] Veure trasparéncia 11/31 del Tema 2 per a recordar tipus valor (o primitius) i tipus referència

6.2.1 Introducció a les taules

- **Concepte de taula**: estructura de dades per a **dades homogènies**, és a dir, per a contenir N elements del mateix tipus. Els elements emmagatzemats s'identifiquen a través d'un **índex de posició** , que comença en 0, dins la taula de valors.

0	1	2	3
'H'	'o'	'l'	'a'

- **`array`**: tipus básic NO primitiu (tipus referència) existent al llenguatge Java que es pot utilitzar per a implementar el concepte de taula (com a alternativa, també existeixen classes Java que es podrien utilitzar, per exemple *ArrayList*, però NO ho farem en aquesta assignatura).

6.2.2 Declaració i inicialització

- Declaració: `<tipus_elements> [] <nom_variable>;`
  - `int [] a;` també s'accepta estil C `int a[];`
  - `String [] b;`
- Inicialització: Un cop inicialitzat el nombre d'elements, el tamany de l'**array** és inalterable!

```
new <tipus_elements>[<tamany>]
  {<elem1>, ...  , <elem3>}
```

  - `int [] a = new int[10];` 10 elements
  - `int [] a = {1,2,3,s.length()};` 4 elements, índexs [0..3]
  - `String [] m ={''hola'',''adéu''.substring(0,2)};`
  - `float [] a;`
    `a = new float [ ] {1.1f, 2.2f};` podem inicialitzar després de declarar.

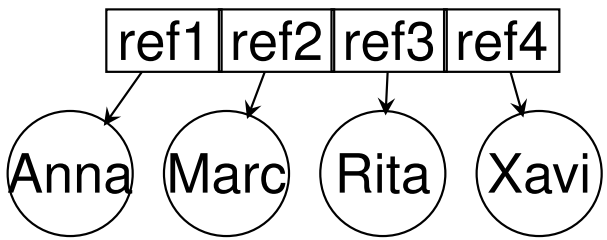
Tipus emmagatzemats

Els arrays poden contenir tipus valor o tipus referència:

- tipus valor:

0	1	2	3
5.2f	7.7f	9.0f	5.0f

- tipus referència:



Tipus emmagatzemats

- Exemples: què es crea a memòria amb el codi següent?

```
String [ ] m = new String[3];

// Assigno, a l'element en la pos. 0,
// la referència al literal ''hola''
m[0] = "hola";

// Assigno, a l'element en la pos. 1,
// la referència al String que n'hi ha a la posició 0
m[1] = m[0];

// Assigno la referència a un substring
// del String al que referència m[1]
m[2] = m[1].substring(0,2);
```

6.2.3 Manipulació del contingut

- **Operacions bàsiques**:
  - **Consulta**: `<id_var>[<idx>]`
  - **Assignació**: `<id_var>[<idx>] = <expressió>`
  - en molts llenguatges, els índexs comencen pel 0 i el límit superior es comprova automàticament.
- **Altres operacions**:
  - **Longitud**: `<id_var>.length` és longitud declarada
  - **Comparació**: `Arrays.equals(<id_var_array1>,<id_var_array2>)`
  - **Còpia**: `System.arraycopy(<src>, <srcPos>, <dst>, <dstPos>, lon)`
  - Funcions de la classe Arrays: `Arrays.equals`, `Arrays.fill`, `Arrays.sort`, `Arrays.binarySearch`...per utilitzar aquestes funcions s'ha de fer *import java.util.Arrays*;

6.2.4 Esquemes de programació de taules

- Podem **interpretar les taules com a seqüències d'elements**, i per tant podem adaptar els *esquemes de programació de seqüències*:
  - recorregut
  - cerca
- **Caracterització de la seqüència** dels índex d'una taula:
  - Primer() = 0
  - Següent(x) = x+1
  - FiSeq(x) = (x $>=$ nombre_elements_taula)

Recorregut I

- Esquema de recorregut en una taula:

```
for(idx=0; idx<taula.length ;idx++){
    <sentències tractar taula[idx]>
}
```

- Més endavant a ProgII fareu ús de colleccions i utilitzareu la sentència `foreach`.

```
import java.util.Scanner;
public class TaulaRecorregut {
    public static final int TAMANY = 10;
    public static void main (String[] args) {
        int [] taula = new int[TAMANY];
        Scanner sc;
        sc = new Scanner(System.in);
        for (int idx=0; idx < TAMANY ; idx++) {
            System.out.println("Valor␣"+(idx+1)+"?␣");
            taula[idx] = sc.nextInt();
        }
        for (int idx=0; idx < taula.length ; idx++) {
            System.out.println("Element␣"+(idx+1)+"␣=␣"+taula[idx]);
```

Recorregut II

```
                }
        }
}
```

Cerca

- Esquema de cerca per taules:

```
idx=0; trobat=false;
while ( (idx<taula.length) and !trobat ){
    if (<condició cerca sobre taula[idx]>){
        trobat=true;
    } else {
        idx++;
    }
}
```

Cerca I

```
import java.util.Scanner;
/* Aquest program cerca un nom en un array de Strings */
public class TaulaCerca {
  public static void main (String[] args) {
    int idx;  boolean trobat; String nom;
    Scanner sc =  new Scanner(System.in);
    String [] taula = {"Oriol", "Eulàlia", "David", "Santi"};

    System.out.println("Qui vols cercar?");
    nom = sc.next();


    /* Penseu el codi que falta aquí */


    if (trobat)
       System.out.println("Trobat a la posició "+(idx+1));
    else
         System.out.println("Nom no trobat");
  }

}
```

Exemple: I

Dígits continguts en els múltiples de 7: donats els múltiples de 7 inferiors a 10000, donar quants dígits 0's, quants dígits 1's, . . . , quants dígits 9 contenen.

- Identificació de la seqüència principal: múltiples de 7 inferiors a 10000
  - Primer element: m= 7
  - Següent element: m = m + 7
  - Final de seqüència: $m > 10000$
  - Identificació de l'esquema: Recorregut

  N'hi han dues seqüències més (veure codi a continuació)

Exemple: I

```
public class ComptarDigitsSenseMetodes {
    public static final int TAMANY = 10;
    public static void main (String[] args) {
        int [] comptador; int m, mult, d;
        comptador = new int[TAMANY];

        for (int i = 0; i < comptador.length; i++)
            comptador[i] = 0;
        /* Identificacio de la sequencia: multiples de 7
            Primer(): m = 7
            Seguent(m):  m = m + 7
            FinalSeq(m): m > 10000
            Identificacio de l'esquema: Recorregut
        */
        m = 7;
        while (m <= 10000) {

           /* Penseu el codi que falta aqui */

          m = m + 7;
        }
```

Exemple: II

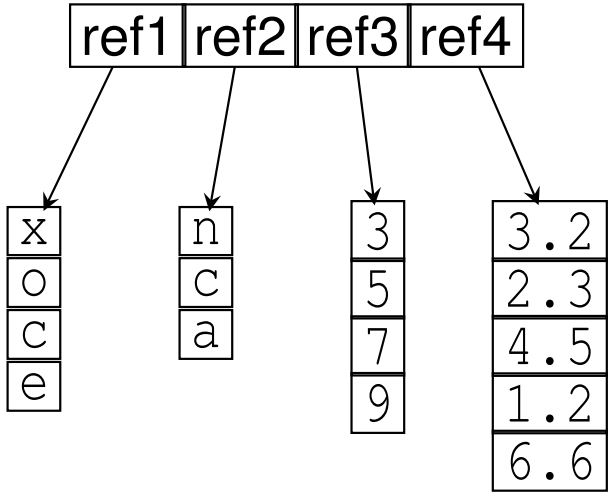
```
        // Imprimir multiples
        for (int i = 0; i < comptador.length; i = i+1) {
            System.out.println ("comptador␣"+i+":␣"+comptador[i]);
        }
    }
}
```

Introducció a les matrius/arrays de vàries dimensions

- **matriu** = taula N dimensional
- Conceptualment són homogènies (de contingut)

21	02	13	45
12	32	43	59
13	52	36	85

- Les podem implementar com una taula que conté altres taules (**aniuament**).
- Podrien ser heterogènies (de contingut i tamany).



Declaració i inicialització de matrius

- Declaració: `<tipus_elements> []<...>[] <id_variable>;`
  - `int [][] m;` declaració dues dimensions
- Inicialització:
  `new <tipus_elem>[<mida>]<...>[<mida_opcional>]`
  `{ {<el1>, ... , <elN>}, <...> , {<el1>, ... , <elN>} }`
  - `int [] [] array1 = new int [3][3];` //matriu 9 elements, sense inicialitzar
  - `int [] [] array2 = { {11, 21}, {12, 22} };` // matriu 4 elements, inicialitzada amb literals
  - `int [] [] array3 = new int [2][];` //només s'especifica la "primera' dimensió
    - `array3[0] = new int [7];`
    - `array3[1] = new int [] {3,6,4,1,8};` //també com a literals, 5 elements

Esquemes de programació de matrius I

- Si les interpretem com a **taules aniuades**, podem adaptar els *esquemes de programació de taules*, fent **aniuament de bucles**.

```
public class MatriuRecorregut {
  public static void main (String[] args) {
    int [][] taula = { {1, 2}, {3, 4} };

    for (int fil=0; fil < taula.length ; fil++) {

      for (int col=0; col < taula[fil].length ; col++) {

        System.out.println("Element␣("+(fil+1)+
                           ",␣"+(col+1)+
                           ")="+taula[fil][col]);
      }
    }
  }
}
```