Qualche nota sugli array (o 'vettori')

Eugenio G. Omodeo

UNIVERSITÀ
DEGLI STUDI DI TRIESTE

Trieste, 27/10/2015

A che servono gli *array* ?

- Ad aggregare componenti. . .

A che servono gli *array* ?

- Ad aggregare componenti. . .
  - . . . di tipo omogeneo,

A che servono gli *array* ?

- Ad aggregare componenti. . .
  - . . . di tipo omogeneo,
  - che possono anche ripetersi,

A CHE SERVONO GLI *array* ?

- Ad aggregare componenti. . .
  - . . . di tipo omogeneo,
  - che possono anche ripetersi,
  - individualmente indicizzate con interi consecutivi

$$0, \ldots, \ell \,,$$

A CHE SERVONO GLI *array* ?

- Ad aggregare componenti. . .
  - . . . di tipo omogeneo,
  - che possono anche ripetersi,
  - individualmente indicizzate con interi consecutivi

$$0, \ldots, \ell \,,$$

  - individualmente modificabili,

A che servono gli *array* ?

- Ad aggregare componenti. . .
  - . . . di tipo omogeneo,
  - che possono anche ripetersi,
  - individualmente indicizzate con interi consecutivi

$$0, \ldots, \ell\,,$$

  - individualmente modificabili,
  - ad "accesso *random*".

A che servono gli *array* ?

- Ad aggregare componenti. . .

- Eventuali ripetizioni delle componenti sono ammesse.

A che servono gli *array* ?

- Ad aggregare componenti. . .

- Eventuali ripetizioni delle componenti sono ammesse.
- 'Conta' l'ordine in cui figurano le componenti.

A che servono gli *array* ?

- Ad aggregare componenti. . .

- Eventuali ripetizioni delle componenti sono ammesse.
- 'Conta' l'ordine in cui figurano le componenti.
- La lunghezza (*length*) non è modificabile ed è finita.

A che servono gli *array* ?

- Ad aggregare componenti. . .

- Eventuali ripetizioni delle componenti sono ammesse.
- 'Conta' l'ordine in cui figurano le componenti.
- La lunghezza (*length*) non è modificabile .
- Il valore **null** è ammesso.

A che servono gli *array* ?

- Ad aggregare componenti. . .

- Eventuali ripetizioni delle componenti sono ammesse.
- 'Conta' l'ordine in cui figurano le componenti.
- La lunghezza (*length*) non è modificabile .
- Il valore **null** è ammesso.
- Esempi ?

. . . . . .

Come si rappresenta una '*matrice*' matematica ?

Come si rappresenta una '*matrice*' matematica ?

Tramite *array* bidimensionale

Come si rappresenta una '*matrice*' matematica ?

Tramite *array* bidimensionale

⟨ nome_array ⟩ **[**⟨ indice_riga ⟩**]** **[**⟨ indice_colonna ⟩**]**

Come si rappresenta una '*matrice*' matematica ?

Tramite *array* bidimensionale

⟨ nome_array ⟩ [⟨ indice_riga ⟩] [⟨ indice_colonna ⟩]

⇒

	0	1	2	3	4	5	6	7
0	T	C	A	Q	R	A	C	T
1	P	P	P	P	P	P	P	P
2								
3								
4								
5								
6	P	P	P	P	P	P	P	P
7	T	C	A	Q	R	A	C	T

La *length* è davvero immodificabile ?

```
int [] vet   ;
       vet   =   new int [10];

       vet   =   new int [0];
       vet   =   null
```

(E allora questo è sbagliato ?)

La *length* è davvero immodificabile ?

```
int [ ] vet    ;
        vet    =    new int [ 10 ];
int [ ] wet    =    vet;
        vet    =    new int [ 0 ];
        vet    =    null
```

(E allora questo è sbagliato ?)

Come stampare un *array* ?

Si prendano a modello i seguenti metodi:

```
public static int sommaCumulativa( int[] numeri ) {

    assert numeri != null;

    int res = 0;

    for ( int j = 0; j < numeri.length; j++ )

        res += numeri[ j ];

    return res;
}

public static int prodCumulativo( int[] numeri ) {
```

Come stampare un *array* ?

Si prendano a modello i seguenti metodi:

```
public static int maxCumulativo( int[] numeri ) {

    assert numeri != null;

    int res = Integer.MIN_VALUE;

    for ( int j = 0; j < numeri.length; j++ )

        res = ( res > numeri[ j ] ) ? res : numeri[ j ];

    return res;
}
```

Come stampare un *array* ?

Si prendano a modello i seguenti metodi:

```
public static boolean tuttiPari( int[] numeri ) {

    assert numeri != null;

    boolean res = true;

    for ( int j = 0; j < numeri.length; j++ )

        res = res && (numeri[ j ] % 2 == 0);

    return res;
}

public static boolean qualcheDispari( int[] numeri ) {
```

Come stampare un *array* ?

Ora siamo pronti per rispondere alla domanda sulla stampa:

```
// Il seguente metodo implementa una comunissima
//                        astrazione sul controllo:

public static String aStampa( int[] numeri ) {

    if ( numeri == null ) return "??";

    String res = "";

    for ( int j = 0; j < numeri.length; j++ )

        res += " " + numeri[ j ];

    return res;
}
```

Come effettuare l'esperimento sulla *length*:

```
class LengthImmod{ // modifico la lunghezza di un array?

    public static void main ( String[] scovasse ){

        int[] vet;

              vet = new int[ 10 ];

        for ( int j = 0; j < vet.length; j++ )

            vet[ j ] = j+1;

        System.out.println( aStampa( vet ) );

        int[] wet = vet;  vet[ 0 ] = 99;  // N.B. !!!

              vet = new int[  0 ];

        System.out.println( aStampa( vet ) );

              vet = null;

        System.out.println( aStampa( vet ) );

        System.out.println( aStampa( wet ) );
```

Come effettuare l'esperimento sulla *length*:

```
Eugenios-MacBook-Air:131022 eomodeo$ javac LengthImmod.java
Eugenios-MacBook-Air:131022 eomodeo$ java LengthImmod
 1 2 3 4 5 6 7 8 9 10

??
 99 2 3 4 5 6 7 8 9 10
SOMMA = 153
PROD = 359251200
MAXX = 99
TUTTI_PARI = false
QUALCHE_DISPARI = true
```

Come effettuare l'esperimento sulla *length*:

```
Eugenios-MacBook-Air:131022 eomodeo$ javac LengthImmod.java
Eugenios-MacBook-Air:131022 eomodeo$ java LengthImmod
 1 2 3 4 5 6 7 8 9 10

??
 99 2 3 4 5 6 7 8 9 10
SOMMA = 153
PROD = 359251200
MAXX = 99
TUTTI_PARI = false
QUALCHE_DISPARI = true
```

Morale: Java ha, per quanto riguarda gli oggetti, una semantica per 'riferimento'.

Come effettuare l'esperimento sulla *length*:

```
Eugenios-MacBook-Air:131022 eomodeo$ javac LengthImmod.java
Eugenios-MacBook-Air:131022 eomodeo$ java LengthImmod
 1 2 3 4 5 6 7 8 9 10

??
 99 2 3 4 5 6 7 8 9 10
SOMMA = 153
PROD = 359251200
MAXX = 99
TUTTI_PARI = false
QUALCHE_DISPARI = true
```

Morale: Java ha, per quanto riguarda gli oggetti, una semantica per 'riferimento'.

Il *garbage collector* smaltirà gli oggetti non piú riferiti...

array: È severo il requisito di omogeneità ?

Come ci ha indicato l'esempio della scacchiera, possiamo formare *array* di *array*...

array: È SEVERO IL REQUISITO DI OMOGENEITÀ ?

Come ci ha indicato l'esempio della scacchiera, possiamo formare *array* di *array*...

...ad es. una matrice rettangolare,

array: È severo il requisito di omogeneità ?

Come ci ha indicato l'esempio della scacchiera, possiamo formare *array* di *array*...

...ad es. una matrice rettangolare,

...che può anche avere componenti di varie lunghezze

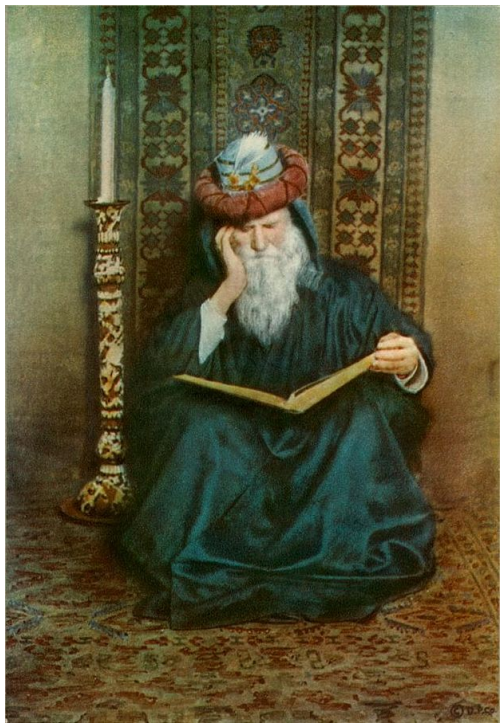
array: È SEVERO IL REQUISITO DI OMOGENEITÀ ?

Come ci ha indicato l'esempio della scacchiera, possiamo formare *array* di *array*...

...ad es. una matrice rettangolare,

...che può anche avere componenti di varie lunghezze

Esempio: Triangolo di Khayyàm / Tartaglia / Pascal



Esercizio sui coefficienti binomiali

Qualche lezione fa abbiamo visto una regola per calcolarli *ricorsivamente*.

Esercizio sui coefficienti binomiali

Qualche lezione fa abbiamo visto una regola per calcolarli *ricorsivamente*.

Avete provato a implementarla in Java ?

Esercizio sui coefficienti binomiali

Qualche lezione fa abbiamo visto una regola per calcolarli *ricorsivamente*.

Avete provato a implementarla in Java ?

Riuscite a fare a meno della ricorsione utilizzando un *array* triangolare ?

Quanto è severo il requisito di omogeneità ?

Come vedremo, i tipi di oggetti formano in classe una gerarchia che comprende tipi piú o meno specifici. Grazie a ciò potremo scrivere:

Quanto è severo il requisito di omogeneità ?

Come vedremo, i tipi di oggetti formano in classe una gerarchia che comprende tipi piú o meno specifici. Grazie a ciò potremo scrivere:

```
class ArrayOggetti{

    public static void main ( String[] scovasse ){

        Object[] vet = new Object[ 2 ];

        vet[ 0 ] = 0;

        vet[ 1 ] = 1.0;

        System.out.println( aStampa( vet ) );
    }

    public static String aStampa( Object[] numeri ) {

        if ( numeri == null ) return "??";

        String res = "";
```

Array a tre dimensioni: dal colore al *b&w*

Le immagini a **livelli di grigio**, anche dette a **scala di grigi**, **grayscale** o **greyscale**, sono immagini digitali raster in bianco e nero costituite da una gamma di grigi (in genere da 4 a 256 livelli o toni) che vanno dal bianco al nero.

Alcuni programmi di grafica permettono di convertire una fotografia a colori in livelli di grigio.

Immagine a colori RGB | Immagine a 256 livelli di grigio

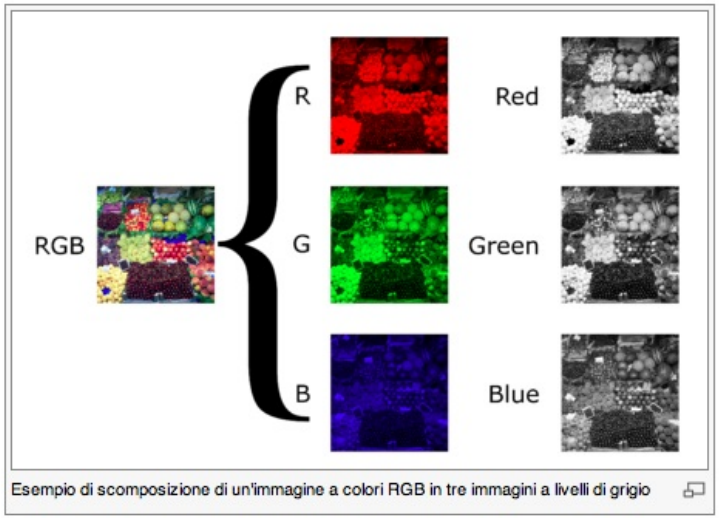
Scala di grigi canali come singole immagini a colori multicanale [modifica | modifica sorgente]

Questa voce o sezione sugli argomenti informatica e fotografia è ritenuta da controllare.
Motivo: *Sezione fuori contesto.*

Partecipa alla discussione e/o correggi la voce. Segui i suggerimenti dei progetti di riferimento 1, 2.

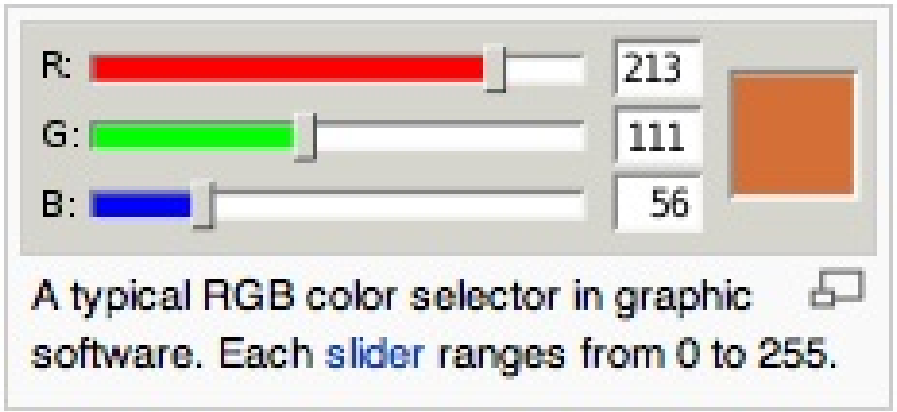
Le immagini a colori sono spesso costruite su diversi canali di colore sovrapposti, ciascuno di loro che rappresa i livelli di valore del canale specificato. Ad esempio, le immagini RGB sono composti da tre canali indipendenti per il rosso, verde e blu, i colori primari, le immagini CMYK hanno quattro canali per ciano, magenta, giallo e nero, ecc...

Ecco un esempio di suddivisione di un canale di colore l'immagine a colori RGB. La colonna a sinistra mostra i canali di colore separati nei colori naturali, mentre a destra si trovano le immagini in scala di grigio equivalenti:



Esempio di scomposizione di un'immagine a colori RGB in tre immagini a livelli di grigio

Come rappresentare il RGB ?



A typical RGB color selector in graphic software. Each slider ranges from 0 to 255.

http://www.javascripter.net/faq/rgbtohex.htm
http://www.javascripter.net/faq/hex2cmyk.htm
http://www.javascripter.net/faq/rgb2cmyk.htm

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Proviamo allora a ragionare direttamente in *bit*:

Un'immagine RGB è un array bidim. di array di ... ... *bit*

Una in tonalità di grigio è un array bidim. di array di ... *bit*

Struttura dell'"ingrigitore" –I

```
final static int tagliaRGB = 24;

final static int tagliaTon =  8;

public static boolean[][][] rgb2tongri( boolean[][][] im ){

    assert im != null && im.length > 0 && im[0].length > 0

                      && im[0][0].length == tagliaRGB;

    boolean[][][] res =

        new boolean[ im.length ][ im[0].length ][ tagliaTon ];

    for( int i = 0; i < im.length; i++ )

        for( int j = 0; j < im[ 0 ].length; i++ )

            res[ i ][ j ] = rgb2tongri( im[ i ][ j ] );

    return res;
}
```

Struttura dell'"ingrigitore" –II

```
class RGB2tongri{ // da RGB a 256 tonalita` di grigio

    ......................
    ......................
    ......................

    private static boolean[] rgb2tongri( boolean[] rgb ){

        assert rgb != null && rgb.length == tagliaRGB;

        boolean[] res = new boolean[ tagliaTon ];

        return res; // questo e` uno stub! non abbiamo convertito nulla!
    }
}
```

(Prima di questo metodo privato, quello pubblico visto sopra)

Esercizio piú semplice, ma dello stesso tipo

Riempi a caso un array di numeri naturali e conta quanti sono i pari.

Riempimento casuale

```
public static int[] scegliACaso( int quanti, int maxIntervallo ) {

    assert quanti > 0 && maxIntervallo > 0;

    int[] wet = new int[ quanti ];

    for ( int j = 0; j < wet.length; j++ )

        wet[ j ] = (int) ( Math.random() * ( maxIntervallo + 1 ) );

    return wet;
}
```