SKILL FACTORY

Scuola **E**sperienza **A**ggiornamento **L**avoro

www.skillfactory.it

SKILL FACTORY

Scuola Esperienza Aggiornamento Lavoro

JAVA – SVILUPPO APPLICAZIONI DESKTOP

Logica di programmazione

Autore: Mirko Onorato – Skill Factory Srl

www.skillfactory.it

Logica di programmazione

autore Mirko Onorato – Skill Factory Srl

L’utilizzo di questo materiale didattico è riservato solo ai partners e gli studenti autorizzati dalla Skill Factory S.r.l., con licenza AUTSFLOPRV01.01. Non può essere usato a fini commerciali e al di fuori di qualunque percorso di formazione non riconosciuto dalla Skill Factory.

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

Gli ***array o vettori*** sono oggetti particolari che fungono da contenitori di dati, sia primitivi che referenziati. Si definiscono come strutture dati statiche in quanto hanno una lunghezza definita dal programmatore e che corrisponde al numero di elementi contenibili nel suo interno. A differenza dei classici oggetti che risultano associati ad una specifica classe, un array vanta della **dichiarazione dinamica**. Un vettore viene infatti dichiarato rispetto il tipo di elementi che esso contiene:

```
int[] vettore1;
float[] vettore2;
Object[] vettore3;
Persona[] vettore4;
```

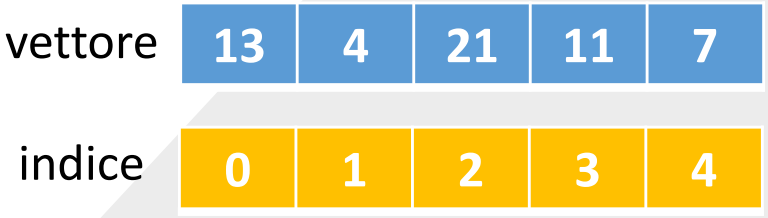
Inoltre, non essendo associato ad una classe specifica, non vanta di alcun costruttore. Istanziare un array significa usare la keyword new come per tutti gli oggetti, ma combinandolo col nome del tipo degli elementi contenuti, specificando la lunghezza dell'array stesso:

```
int[] vettore1=new int[n]; //n è la lunghezza dell'array
float[] vettore2=new float[n];
Object[] vettore3=new Object[n];
Persona[] vettore4=new Persona[n];
```

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

Un array è strutturato in celle, ognuna delle quali contiene l'i-esimo dato ed è individuata da un indice. L'indice è quindi la posizione del singolo elemento e assume valori che vanno da 0 (prima posizione) a n-1 (ultima posizione), ove n è la dimensione dell'array e quindi il numero di dati contenuti in esso.

```
int[] vettore=new int[5]; //il vettore contiene 5 numeri interi
vettore[0]=13;
vettore[1]=4;
vettore[2]=21;
vettore[3]=11;
vettore[4]=7;
```



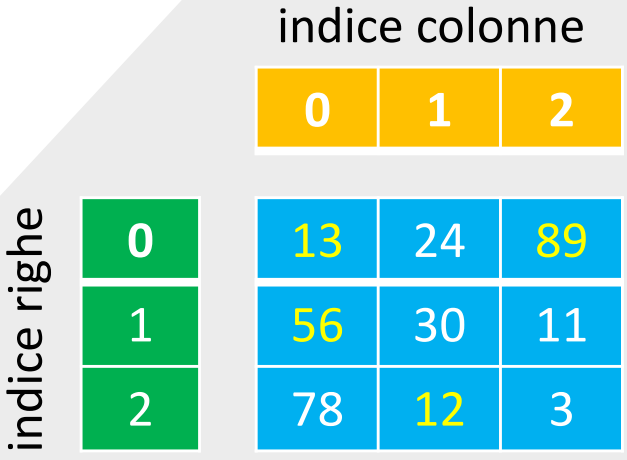
In alternativa, è possibile inizializzare un array in modo più rapido:

```
int[] vettore={13,4,21,11,7};
```

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

Gli array si dicono monodimensionali quando presentano una sola dimensione; di contro una matrice è un **array multidimensionale** in quanto si avvale di due dimensioni: una relativa alle righe ed una alle colonne:

```
//Matrice nxm, n righe x m colonne
int[][] matrix=new int[n][m];
matrix[0][0]=13;
…
matrix[0][2]=89;
matrix[1][0]=56;
…
matrix[2][1]=12;
…
```



2.1 – TECNICA AVANZATA: VETTORI E MATRICI

```
//Cerca il minimo ed il massimo in un vettore di interi
public void cercaMinimoEMassimo() {
int[] v = {2,7,5,6,8,1,4,3,9};
int min = v[0];
int max = v[0];
//Ricerca del minimo
for(int i=1; i<v.length; i++) {
if(min > v[i])
min=v[i];
}
//Ricerca del massimo
for(int i=1; i<v.length; i++) {
if(max < v[i])
max=v[i];
}
System.out.println("Il minimo è: "+min);
System.out.println("Il massimo è: "+max);
}
```

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

```
//Fai la somma e la media del contenuto di un vettore di interi
public void sommaEmedia() {
int[] v = {2,7,5,6,8,1,4,3,9};
int totalizzatore=0;
for(int i=0; i<v.length; i++) {
totalizzatore=totalizzatore+v[i]; //in alternativa: totalizzatore+=v[i];
}
System.out.println("La somma è pari a: "+totalizzatore);
System.out.println("La media è pari a: "+totalizzatore/v.length);
}

//Visualizza i valori di un vettore di interi al contrario
public void stampaAritroso() {
int[] v = {2,7,5,6,8,1,4,3,9};
for (int i=v.length-1; i>=0; i--) {
System.out.print(v[i]+" ");
}
}
```

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

```
//Somma a due a due otto interi di un vettore, indicando l'i-esima coppia
public void sommaAdueAdue() {
int[] v = {2,7,5,6,8,1,4,3};
int nCoppia=1;
for (int i=0; i<v.length; i++) {
System.out.print("Coppia n."+nCoppia+"\n Somma:"+(v[i]+v[i+1])+"\n");
i++;
nCoppia++;
}
}
```

```
Coppia n.1
 Somma:9
Coppia n.2
 Somma:11
Coppia n.3
 Somma:9
Coppia n.4
 Somma:7
```

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

```
//Fare una matrice 10x10 ove ogni riga/colonna rappresenta una tabellina
public void tabelline() {
int[][] tabelline = new int[10][10];
for(int i=0; i<10; i++) {
System.out.println("");
for(int j=0; j<10; j++) {
tabelline[i][j]=(i+1)*(j+1);
System.out.print(tabelline[i][j]+" ");
}
}
}
```

```
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

2.1 – TECNICA AVANZATA: VETTORI E MATRICI

```
//Creare una matrice 'meteo' 8x5 con i campi: Giorno [h]:0-6 [h]:7-13 [h]:14-19 [h]:20-0
public void meteo() {
int[][] matrix = new int[7][4];
String [] campi = {"Giorno", "[0-6]", "[7-13]", "[14-19]", "[20-0]"};
String [] giorni = {"Lunedì","Martedì","Mercoledì","Giovedì","Venerdì","Sabato","Domenica"};
for(int x=0; x<=4; x++) {
System.out.print(campi[x] + " ");
}
System.out.println();
for(int righe=0; righe < 7; righe++) {
System.out.print(giorni[righe] + " ");
for(int colonne= 0; colonne < 4; colonne++) {
matrix[righe][colonne] = (int) (Math.random()*35);
System.out.print(matrix[righe][colonne]+ " ");
}
System.out.println(" ");
}
}
```

```
Giorno [0-6] [7-13] [14-19] [20-0]
Lunedì 16 33 18 27
Martedì 22 9 3 27
Mercoledì 22 1 12 17
Giovedì 15 31 33 0
Venerdì 5 15 13 3
Sabato 19 29 24 31
Domenica 4 34 15 9
```

2.2 – TECNICA AVANZATA: ORDINAMENTO

L'ordinamento (il sorting) nasce con l'esigenza di organizzare il contenuto di una struttura dati (es.un vettore) rispetto dei criteri; nel caso dei numeri l'ordinamento consiste nel disporli in ordine crescente o decrescente, mentre con le stringhe si ragiona in termini di lunghezza o numero dei caratteri, crescente o decrescente. Esistono vari algoritmi utili allo scopo tra cui i più usati: ***Selection Sort*** e ***Bubble Sort***.

```
public void selectionSort(int[] array) {
    for(int i = 0; i < array.length-1; i++) {//il lenght-1 indica il numero di scambi
        int minimo = i;
        for(int j = i+1; j < array.length; j++) {
        //Se l'elemento «selezionato» è più piccolo del minimo corrente
        //diventa il nuovo minimo
         if(array[minimo]>array[j]) {
                minimo = j;
            }
        }
        //Se il minimo è diverso dall'elemento di partenza allora avviene lo scambio
        if(minimo!=i) {
        int k = array[minimo];
        array[minimo]= array[i];
        array[i] = k;
        }
    }
}
```

2.2 – TECNICA AVANZATA: ORDINAMENTO

```
public void bubbleSort(int[] array) {
    for(int i = 0; i < array.length; i++) {
        boolean flag = false;
        for(int j = 0; j < array.length-1; j++) {
        //Se l' elemento j e maggiore del successivo allora scambiamo i valori
            if(array[j]>array[j+1]) {
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
                flag=true; //Lo setto a true per indicare che é avvenuto uno scambio
            }
        }
        if(!flag) break; //Se flag=false allora vuol dire che nell' ultima iterazione
                         //non ci sono stati scambi, quindi il metodo può terminare
                         //poiché l' array risulta ordinato
    }
}
```

2.3 – TECNICA AVANZATA: RICERCA SEQUENZIALE O LINEARE

Successivo all'ordinamento segue normalmente la ricerca all'interno di una struttura dati. L'algoritmo più semplice è quello di ***ricerca lineare***, in quanto, indipendentemente da come sono disposti gli elementi, troviamo quello che ci serve cercandolo in maniera diretta. La ***chiave*** è esattamente il valore di confronto con cui rileviamo l'obiettivo.

```
public static void ricercaSequenziale(int[] array , int chiave) {
boolean flag=true;
     for (int i=0; i<array.length; i++) {
          if (array[i]==chiave) {
          flag=false;
              System.out.println("L'elemento trovato è:"+array[i]);
          }
     }
     if(flag)
     System.out.println("L'elemento non è stato trovato");
 }
```

2.4 – TECNICA AVANZATA: RICERCA BINARIA O DICOTOMICA

La ricerca binaria o dicotomica è una tecnica di ricerca più performante di quella sequenziale, in quanto dopo ogni lettura riduce del 50% il numero di elementi del vettore. Recupera non l'elemento cercato ma la sua posizione. La ricerca binaria richiede che gli elementi del vettore siano ordinati in modo crescente.

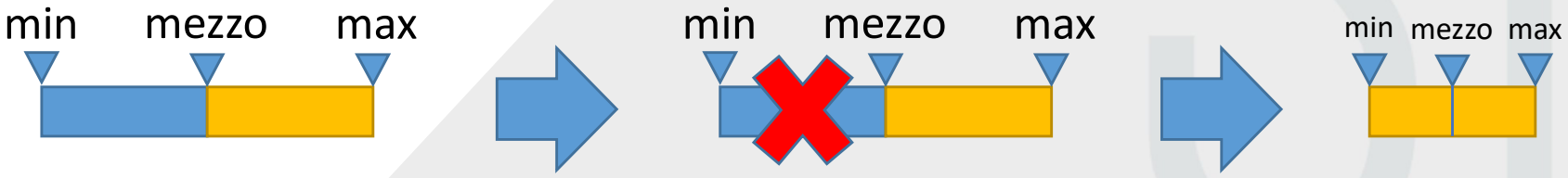
```
public static void ricercaDicotomica (int[] array, int chiave) {
    int min = 0;
    int max = array.length-1;
    int mezzo = 0;
    boolean trovato = false;
    while(!trovato && min <= max) {
        mezzo = (max+min)/2;
        if(array[mezzo] == chiave)
        trovato=true;
        if(array[mezzo] < chiave)
        min = mezzo + 1;
        else
        max = mezzo - 1;
    }
    if(trovato)
    System.out.println("La posizione dell'elemento trovato è pari a:"+mezzo);
    else
    System.out.println("Elemento non trovato");
}
```

2.4 – TECNICA AVANZATA: RICERCA BINARIA O DICOTOMICA

Concentriamo sul cuore dell'algoritmo che è il contenuto del ciclo while:

```
mezzo = (max+min)/2;
if(array[mezzo] == chiave)
trovato=true;
if(array[mezzo] < chiave)
min = mezzo + 1;
else
max = mezzo - 1;
```

Se l'elemento centrale dell'insieme di valori è esattamente l'elemento cercato, l'algoritmo termina; se l'elemento centrale è minore di quello cercato, si elimina il dominio alla sinistra del centro ed il nuovo estremo inferiore è pari a quest'ultimo +1:



Analogamente, si elimina il dominio alla destra del centro se il valore qui posizionato è maggiore di quello cercato; il nuovo estremo superiore scala di 1 rispetto il centro. Il nuovo centro è quello del nuovo dominio.

2.5 – TECNICA AVANZATA: ROTTURA DI CODICE

La Rottura di codice è un algoritmo che permette di contare o sommare i valori uguali, chiamati anche chiavi, di una sequenza. Per applicare la rottura di codice è necessario che la sequenza sia ordinata in modo crescente. Quando il valore della chiave corrente cambia, ***si verifica la rottura di codice***, in questo caso, prima di procedere con i calcoli per la chiave seguente, si stampano i valori della chiave precedente.

```
public static void rotturaDiCodice(int[] array) {
boolean flag;
int somma=0;
int indice=0;
int valore=0;
int iterazione=0;
do {
flag=false;
somma=array[indice];
valore=array[indice];
…
//continua
```

2.5 – TECNICA AVANZATA: ROTTURA DI CODICE

```
//continuazione
…
for(int i=indice+1; i<array.length; i++) {
if(valore==array[i])
somma+=array[i];
else {
flag=true;
iterazione++;
System.out.println("La "+iterazione+"° somma vale:"+somma);
indice=i;
break;
}
}
}
while(flag);
//Ultima somma
iterazione++;
System.out.println("La "+iterazione+"° somma vale:"+somma);
}
```

Se l'array letto è il seguente: `int[] array = {15,15,21,21,34,34};`

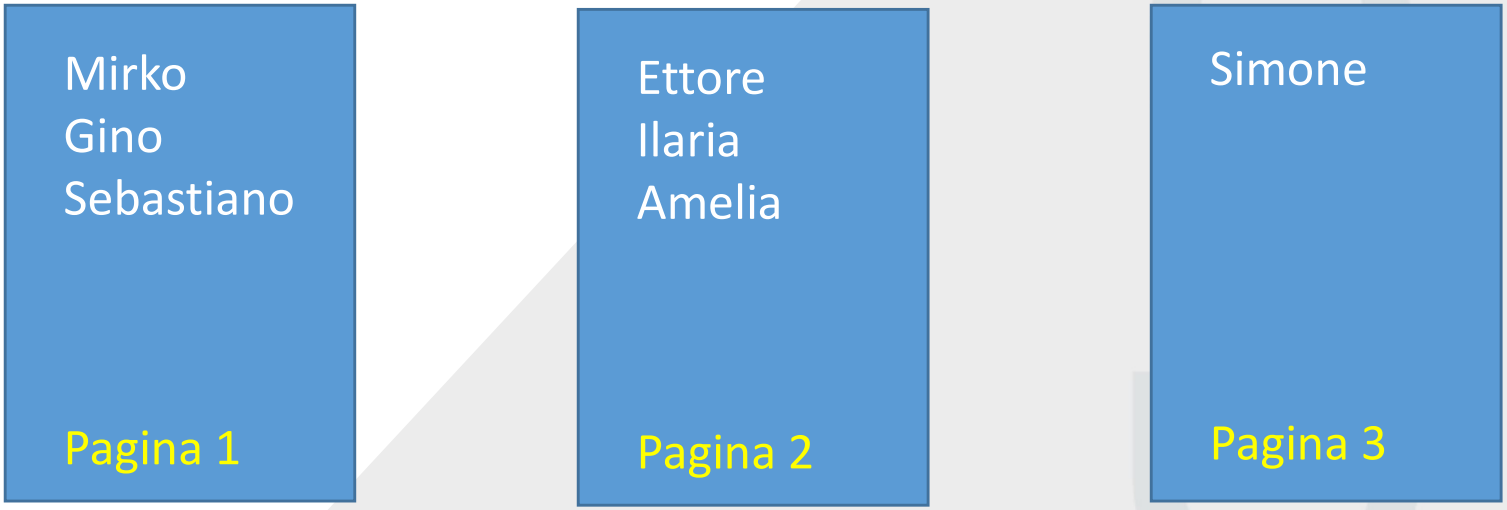
```
La 1° somma vale:30
La 2° somma vale:42
La 3° somma vale:68
```

2.6 – IMPAGINAZIONE

L'impaginazione è l'algoritmo che consente di visualizzare in pagine il contenuto di una struttura dati. Ammettiamo di dichiarare un vettore di stringhe con sette nominativi e di voler visualizzare tre di questi in ogni pagina:

```
String[] array = {"Mirko","Gino","Sebastiano","Ettore","Ilaria","Amelia","Simone"};
```

Avremo un numero di 3 pagine:



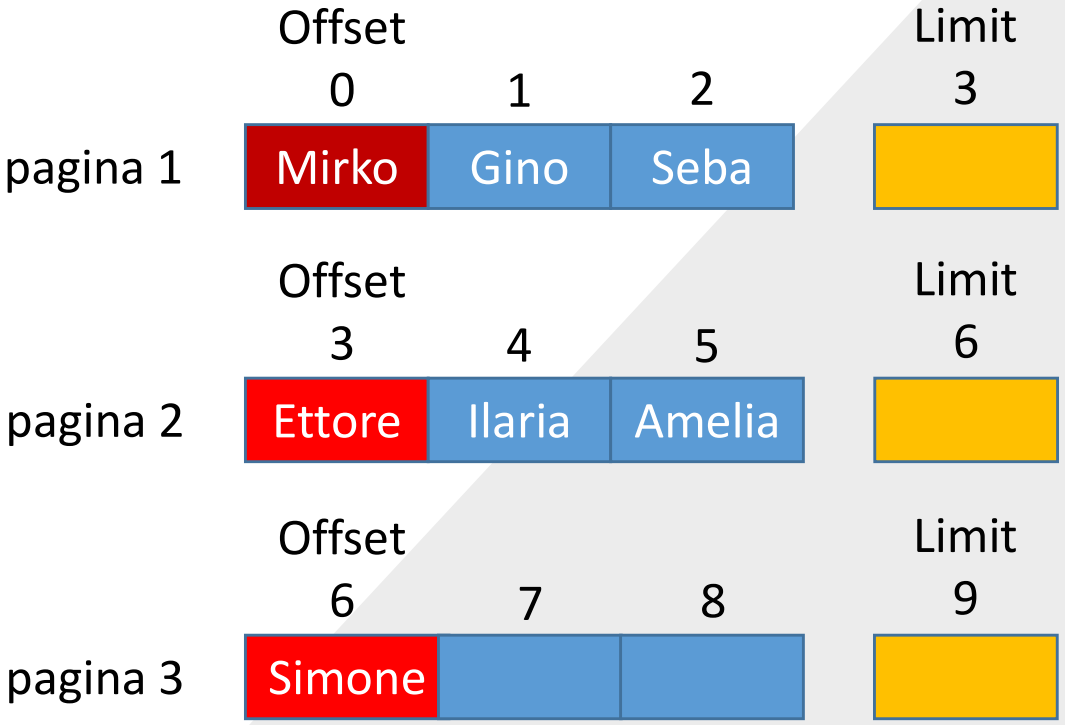
ricavabile dalla seguente formula:

$$nPagine=nElementi/nVisualizzati=7/3=2,33\approx3$$

2.6 – IMPAGINAZIONE

nPagine=nElementi/nVisualizzati=7/3=2,33≈3

L'espressione di cui sopra vede un arrotondamento per eccesso; questo ha luogo quando il rapporto comporta un resto e ciò andrà tenuto in conto nell'algoritmo. Una volta definito il numero di pagine, bisogna definire un limite inferiore ed uno superiore, per riparire gli elementi in pagine. Il limite inferiore è detto **offset** e corrisponde al primo elemento di ogni pagina; di contro, il limite superiore è detto **limit** e corrisponde all'offset della pagina successiva:



offset=(pagina-1)*nVisualizzati

limit=offset+nVisualizzati

2.6 – IMPAGINAZIONE

Per consentire la navigazione tra le pagine è necessaria la presenza di un menu con le seguenti opzioni:

- Vai alla prima pagina
- Vai alla pagina precedente
- Vai alla pagina successiva
- Vai all'ultima pagina

Scegliere una di queste opzioni comporta l'aggiornamento della variabile **pagina**, da cui dipendono i valori di offset e limit.

offset=(**pagina**-1)*nVisualizzati

limit=offset+nVisualizzati

L'opzione che porta alla pagina precedente varrà il decremento **pagina--**, mentre quella relativa alla pagina successiva varrà **pagina++**.

2.6 – IMPAGINAZIONE

```
public static void impaginazione(String[] array) {
Scanner input = new Scanner(System.in);
int pagina=1;
int nPagine=0; //numero delle pagine
int totElementi=array.length; //numero degli elementi nell'array
int nVisualizzati=3; //elementi visualizzati per pagina
int offset;
int limit;
boolean flag=true; //flag che gestisce il do-while
int scelta;
do {
if(totElementi%nVisualizzati!=0) //controlliamo se il rapporto ha resto
nPagine=(totElementi/nVisualizzati)+1; //incremento per eccesso
else
nPagine=(totElementi/nVisualizzati);
if(pagina==0) { //se pagina-- porta a pagina=0, la variabile pagina viene riportata a 1
pagina=1;
}
else if(pagina>nPagine) { //controllo se pagina++ porta a superare il limite delle pagine
pagina=nPagine;
}
else if(pagina==1) {
pagina=1;
}
//continua
```

2.6 – IMPAGINAZIONE

```
//continuazione
offset=(pagina-1)*nVisualizzati;
limit=offset+nVisualizzati;
for(int i=offset; i<limit && i<array.length; i++) {
System.out.println(array[i]);
}
System.out.println("****MENU****"+"\n"+
"1)Prima pagina \n"+
"2)Pagina precedente \n"+
"3)Pagina successiva \n"+
"4)Ultima pagina \n"+
"5)Esci");
scelta=Integer.parseInt(input.nextLine());
//continua
```

2.6 – IMPAGINAZIONE

```
//continua
switch(scelta) {
  case 1:
         pagina=1;
         break;
  case 2:
         pagina--;
         break;
  case 3:
         pagina++;
         break;
  case 4:
         pagina=nPagine;
         break;
  case 5:
         flag=false;
         break;
  default:
         System.out.println("Scegliere un’opzione valida");
  }
}
while(flag);

}
```


THANK
for watching
YOU
www.skillfactory.it