

POO Corso A - Proprietà degli array nativi di Java

`int[] v=new int[5];` `v.length==5`, indici da 0 a 4; `length==capacità`

Possibilità di usare la **notazione con indice** per accedere agli elementi:

`v[0]=4; v[1]=1; v[2]=6; v[3]=-5; v[4]=12;`

...

`v[2]=v[2]+v[0];`

0	1	2	3	4
4	10	6	-5	12

Facilità di trovare successore e predecessore, ammesso che esistano, di un elemento in posizione `i`, `v[i]`:

`v[i]`

`v[i+1]`, se `i<v.length-1`

`v[i-1]`, se `i>0`

Possibilità di ordinare l'array così da supportare la ricerca binaria, etc.

Spesso l'array è incompleto, ossia contiene **size** elementi (indici da 0 a `size-1`). Così, per aggiungere un elemento alla fine:

`int size=2; //dimensione effettiva`

0	1	2	3	4
14	6	?	?	?

 size=2

Posto che `size<length-1`, è sufficiente:

`v[size]=18;`

`size++;`

0	1	2	3	4
14	6	18	?	?


 size=3

`size` punta al **primo buco libero**. Quando `size` diventa uguale a `v.length`, l'array è pieno. Se `size=0`, l'array è vuoto. Semplice, no?

Proprietà che mettono in crisi l'array nativo

L'array è una struttura compatta (elementi incollati uno all'altro). Ammesso che $size < v.length - 1$, che succede volendo inserire un elemento $x=19$ in posizione $i=1$, senza sovrascrittura del 6?

0	1	2	3	4
14	6	18	?	?




occorre shiftare gli elementi dalla posizione i in avanti di un posto (scorrimento destro), quindi si può inserire x :

0	1	2	3	4
14	19	6	18	?

ovviamente $size$ ora è 4.

Dualmente, se si vuole togliere l'elemento esempio in posizione $i=0$, non è sufficiente assegnargli 0 (ingenuità): è necessario fare uno shift a sinistra degli elementi da $i+1$ a $size-1$, così l'elemento precedente $v[0]=14$ scompare, e la $size$ diminuisce di 1:

0	1	2	3	4
14	19	6	18	?



0	1	2	3	4
19	6	18	18	?

Size ora è 3.

Che succede se al tempo di un inserimento (**senza sovrascrittura**) $size$ è già $v.length$? L'inserimento non si può fare a meno di non espandere prima l'array...

Si capisce che inserimenti e rimozioni da una qualsiasi posizione dell'array possono ostacolare l'uso di un array nativo, che rimane di facile uso sino a che es. l'array lo si riempie una volta per tutte e poi ad es. si fanno ricerche, eventualmente ricerca binaria se l'array è ordinato, o si modificano singoli elementi con sovrascrittura etc.

Progetto di un tipo di dati astratto (ADT=Abstract Data Type) Vector (array nativo più flessibile)

Si desidera che siano sempre possibili l'inserimento di un elemento o la rimozione di un elemento in/da qualsiasi posizione, con espansione/contrazione automatica del vettore.

Per far questo si può introdurre un tipo astratto **Vector** (interfaccia) che nasconde logicamente un array nativo al suo interno (un po' come fa String), e costringe a rinunciare all'uso delle [e] per accedere agli elementi: l'accesso ora si fa solo con metodi tipo get/set.

Per generalità (e "massimo" riuso), è conveniente progettare un Vector i cui elementi siano tipo Object (tipo **grezzo**).

```
package poo.util;
public interface Vector{
    int size();                //dimensione effettiva
    int indexOf( Object elem ); //ritorna l'indice della prima occ di elem o -1
    boolean contains( Object elem ); //ritorna true se elem è presente
    Object get( int indice );    //ritorna l'oggetto in posizione indice
    Object set( int indice, Object elem ); //cambia l'oggetto in pos indice e ritorna il prec
    void add( Object elem );    //aggiunge elem alla fine
    void add( int indice, Object elem ); //aggiunge elem in indice (non sovrascrittura)
    void remove( Object elem ); //rimuove la prima occorrenza di elem, se c'è
    Object remove( int indice ); //rimuove e ritorna l'oggetto in pos indice
    void clear();              //svuota il vector
    boolean isEmpty();         //ritorna true se il vector è vuoto
    Vector subVector( int da, int a );
    //ritorna un subvector con gli elementi dalla pos «da» alla posizione «a», a escluso
} //Vector
```

Sono possibili varie eccezioni (unchecked). Es. il metodo get(**indice**) solleva una IndexOutOfBoundsException se **indice** è fuori dall'intervallo 0..size()-1. Similmente per set(...). Il metodo add(**e**) aggiunge **e** alla fine. Se size è uguale alla capacità dell'array, l'array è prima espanso, quindi si fa la add. Similmente per l'altra add, che richiede che l'indice sia nell'intervallo 0..size(). La remove(**indice**) pretende che **indice** sia nell'intervallo 0..size()-1. Vedere codice Java nel package poo.util.