Strutture Dati Astratte

Un piccolo ripasso da Programmazione 1

- La programmazione moderna “ad oggetti” si basa sull’utilizzo di “blocchi” di programmi che racchiudono dati e metodi per la elaborazione degli stessi dati.

- L’idea di “costruire” ***tipi di dati*** e di ***definire in maniera rigorosa operazioni tra tali nuovi tipi di dati*** è una delle idee classiche dell’Informatica.

- Concetti di “**encapsulation**”, “**reuse**” e “**information hiding**”.

- L’obiettivo è quello di raggiungere la massima astrazione nel descrivere i programmi concentrandosi sempre sul “**COSA**” e lasciando al momento della implementazione il “**COME**”.

Abstract Data Type (ADT)

Un ADT è un modello matematico con una collezione di operatori definiti su tale modello.

Più precisamente un ADT si compone di:

- Una collezione di informazioni (**dominio**) omogenee M;
- Alcune funzioni (**operatori**) definiti come

$$f: X \times Y \rightarrow Z$$

ove almeno uno degli insiemi X, Y o Z sia eguale ad M.

Abstract Data Type (ADT)

- Un ***nuovo tipo di dato*** per cui siano stati specificati ***valori*** e ***operazioni possibili*** è detto è detto tipo di dato astratto o semplicemente ADT (Abstract Data Type).

- Lo stesso concetto può essere chiamato anche ***classe*** o ***modulo***.

- Una ragione importante per usare gli ADT in un programma è che tramite essi i dati che costituiscono il tipo di dato astratto sono accessibili solo attraverso le operazioni dell'ADT stesso.

Abstract Data Type (ADT)

- Tale restrizione costituisce una forma di programmazione che mette al sicuro da alterazioni accidentali dei dati provocati da qualche procedura che manipoli i dati in modo non previsto.

- Consente di nascondere il funzionamento interno di una parte di un programma, in modo da proteggere le altre parti del programma dai cambiamenti che si produrrebbero in esse nel caso che questo funzionamento fosse difettoso, oppure si decidesse di implementarlo in modo diverso (incapsulamento).

- Gli ADT consentono di riprogettare le strutture dati e le procedure che realizzano le operazioni su di esse senza la preoccupazione di introdurre errori nelle parti restanti del programma.

- ADT ed I linguaggi OO
  - Obbligano ad *incapsulare* le strutture dati all'interno di opportune dichiarazioni (classi)
  - consentono facilmente di impedire l'accesso alla rappresentazione e all'implementazione (*information hiding*)

Esempi: ADT elementari predefinite nei vari linguaggi di programmazione

- Il tipo **int** di JAVA: M={-2^{16},…,$2^{16}-1$} e le operazioni sono: +,-,/,%,*.
- Il tipo **boolean** di JAVA: M={true, false} e le operazioni sono &&, ||, !.
- Idem per gli altri tipi già visti

Ancora una ADT già nota

ADT = elenco di lunghezza fissata di oggetti cioè un **ARRAY**!

Quali operazioni su un array?

- inserimento di oggetti in un posto dell'elenco determinato da un indice;
- lettura di oggetti da un posto dell'elenco determinato da un indice;
- interrogazione dell'oggetto per conoscere la lunghezza dell'array.

Altro esempio: la classe **String**…

Oltre gli ADT predefiniti

I **tipi elementari o predefiniti** ci consentono di fare molte operazioni ma non esauriscono le nostre necessità di programmazione.

Potere “costruire” noi stessi gli ADT necessari al nostro lavoro e che meglio si adattano a descrivere le informazioni di cui il nostro programma si occupa è importante, comodo e conduce alla costruzione di programmi efficienti, facili da scrivere, *debuggare* e mantenere.

Alcuni ADT sono molto ricorrenti e JAVA predispone alcune classi già pronte che le implementano.

Il nostro obiettivo:

impareremo come sono implemetate gli ADT più comuni e nel processo acquisiremo l’abitudine a pensare in termini di ADT e di loro implementazione.

Non è difficile partendo dagli oggetti!

Un semplice ADT: i numeri complessi

Modello matematico: C="coppie ordinate di numeri reali"
Operazioni (un possibile insieme):
somma, prodotto: C x C→C;
inverso, coniugato: C*→C;
modulo, anomalia, parte reale, parte immaginaria: C→R.

Comunque si implementino queste operazioni in una classe *complex* a chi deve utilizzare tale classe interessa solo COSA fanno le operazioni e non come le realizzano.
Gli **oggetti** complex infatti nasconderanno i dettagli implementativi.
Vedi codice di esempio **esempioADT01.java;**

** l'inverso è definito solo per numeri non nulli*

Un semplice ADT: i numeri razionali

Modello matematico: Q="coppie ordinate di numeri interi, il cui secondo elemento non è mai zero"

Operazioni (un possibile insieme):

somma, prodotto, divisione: Q x Q→Q;

inverso: Q*→C;

riduzione ai minimi termini: Q→Q;

confronto di eguaglianza: Q x Q→{true, false};

Comunque si implementino queste operazioni in una classe razionale a chi deve utilizzare tale classe interessa solo COSA fanno le operazioni e non come le realizzano.
Gli **oggetti** *razionale* infatti nasconderanno i dettagli implementativi.
Vedi codice di esempio **esempioADT02.java;**

** l'inverso è definito solo per numeri razionali non nulli*

Un semplice ADT: i sottoinsiemi di {1…8}

Modello matematico: P = tutti i possibili sottoinsiemi dell'insieme dei primi 8 interi a partire da 1.

Operazioni (un possibile insieme):

unione, intersezione: P x P→P;

inserimento: Px{1..8}→P;

Comunque si implementino queste operazioni in una classe a chi deve utilizzare tale classe interessa solo COSA fanno le operazioni e non come le realizzano. Gli **oggetti** *insieme8* infatti nasconderanno i dettagli implementativi. Vedi codice di esempio **esempioADT03.java;**

Si osservi la severa limitazione a cui siamo soggetti: l'universo da cui estrarre i sottoinsiemi è fisso e non è possibile alterarlo se non intervenendo sul codice e ri-compilando… (ci occorreranno strutture "dinamiche" per superare questo limite).

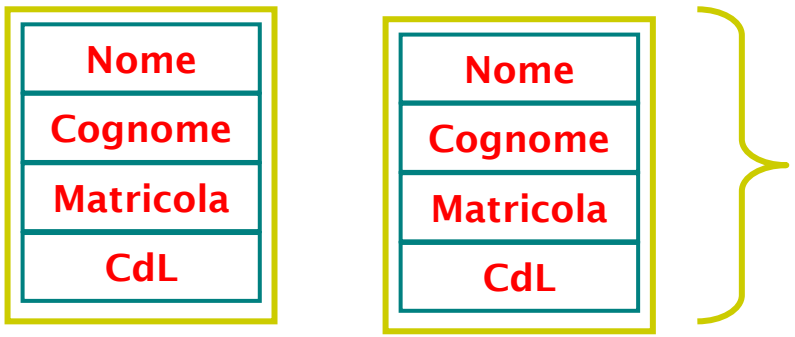
ADT, tipo di dato e struttura dati: chiariamo la differenza tra parole simili

- **ADT:** un modello matematico e operazioni definite su di esso. Definito senza pensare a come esso viene rappresentato e realizzato dentro il computer (in “astratto”);
- **Tipo di dato:** un insieme di possibili valori che una variabile può assumere dentro un programma.
- **Struttura dati:** i modi in cui i tipi di dati sono concretamente organizzati, collegati e messi assieme.

ADT per Organizzare insiemi di dati

Molto spesso bisogna gestire insiemi di oggetti dello stesso tipo (dette liste). Le tipiche operazioni che si vorrebbero eseguire su tali dati sono ad esempio:

- **Inserimento (ordinato, all'inizio, alla fine, …)**
- **Ricerca**
- **Cancellazione**
- **….**



Una serie di importanti domande…

- ***L'elenco delle operazioni sulle liste nella slide precedente è "esaustivo"?*** Esso prevede **TUTTE** le operazioni utili su una lista? E' facile rispondere di no.

- ***L'elenco delle operazioni sulle liste nella slide precedente è "completo"?*** Si possono ottenere da tale elenco mediante combinazione e composizione **TUTTE** le operazioni utili su una lista?

- ***L'elenco delle operazioni sulle liste nella slide precedente è "minimale"?*** In altre parole posso ottenere alcuni degli operatori in elenco come combinazione degli altri?

Le domande di cui sopra sono essenziali per un INFORMATICO: se posso determinare un insieme "completo e minimale" di operazioni necessarie per una particolare applicazione il lavoro di progettazione/programmazione è molto semplificato.

Una serie di importanti domande…

La risposta alle domande posta sopra è spesso assai complessa. Essa dipende da cosa si intende per "TUTTE le operazioni utili".

Altro aspetto da esplorare: un insieme minimale e completo di operatori semplifica la programmazione ma garantisce la massima **efficienza** al nostro programma? In questo caso la risposta non si può ottenere se non guardando con attenzione ANCHE la implementazione che si può ottenere della ADT.

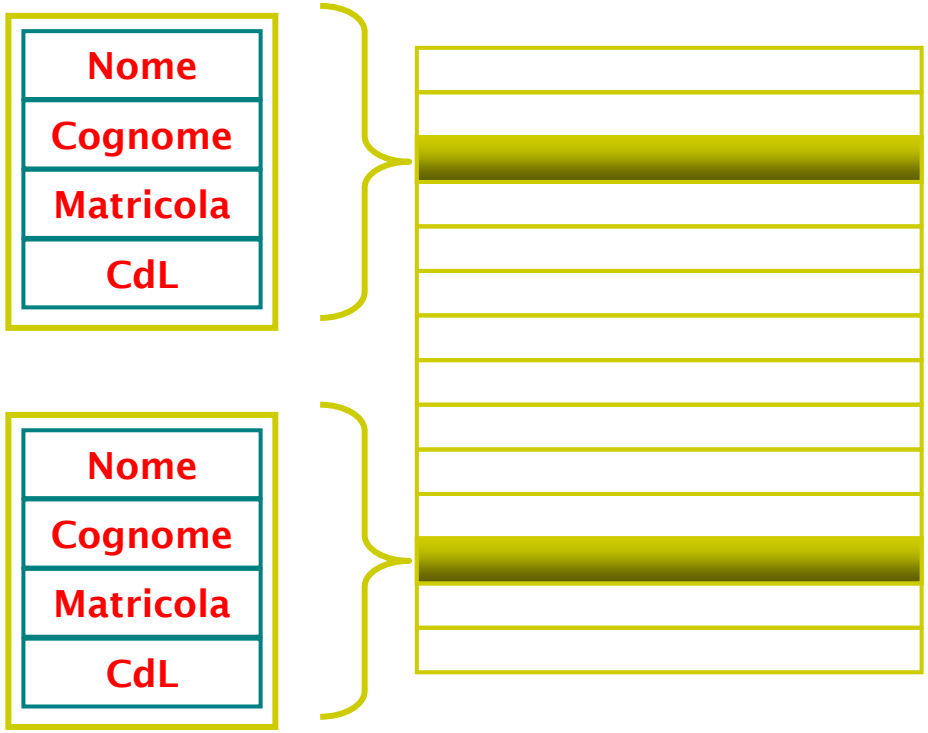
*SAPERE TROVARE LE RISPOSTE A QUESTE DOMANDE NELLE APPLICAZIONI PIU' COMUNI E' IL GRADO DI COMPETENZA RELATIVO ALLE ADT AL QUALE QUESTO CORSO VUOLE CONDURVI (**e che si vuole verificare agli esami**).*

La nostra ADT è (per ora) una lista dei desideri…

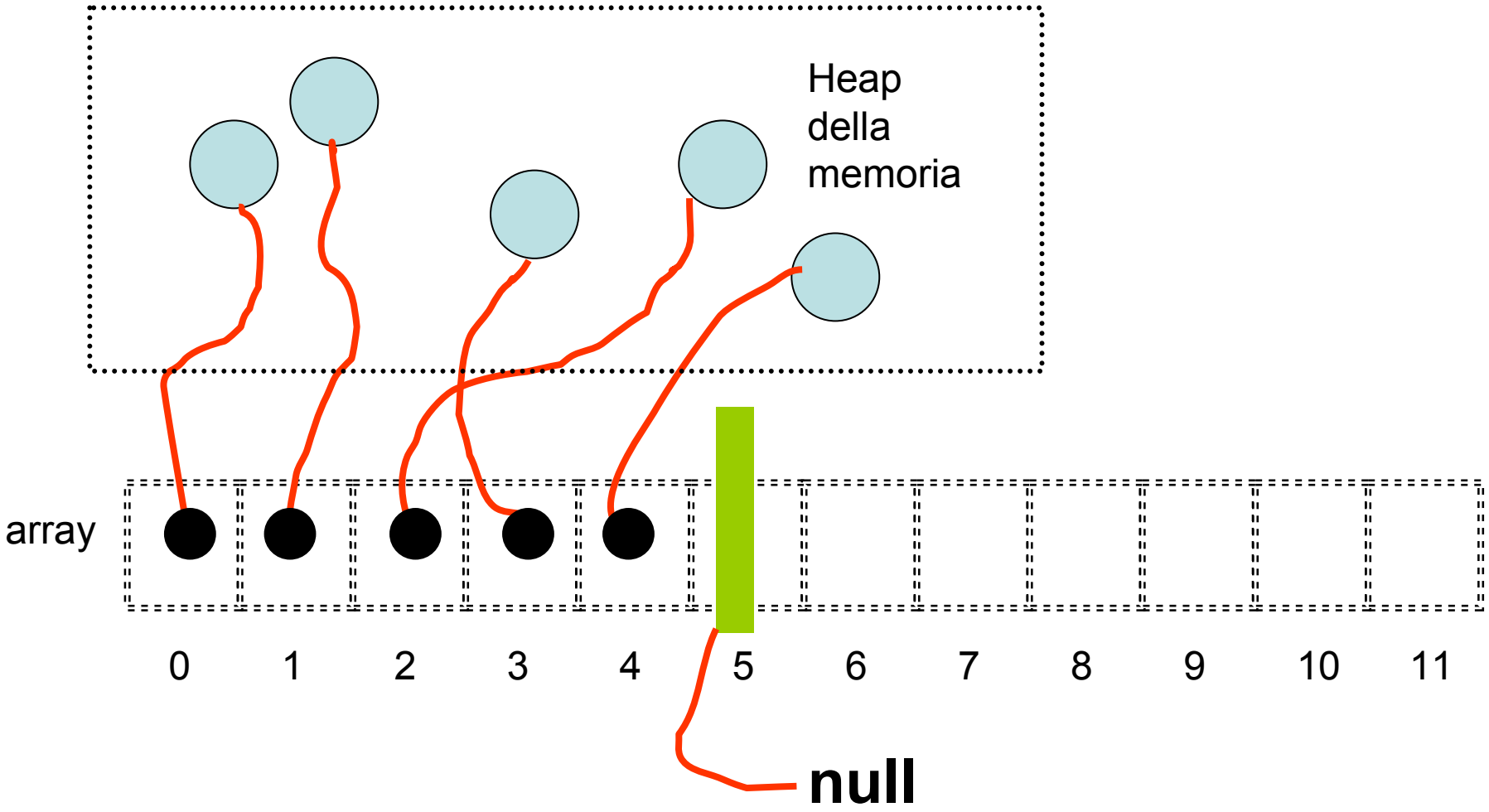
- Come rappresentare una lista secondo la sua definizione come ADT in un programma JAVA?
- Quale delle molte rappresentazioni che vengono in mente rendono facili le operazioni che desideriamo implementare?
- *Se riusciamo però a scrivere una "classe" che garantisca la corretta rappresentazione della lista e di tutte le tipiche (e utili) operazioni, siamo in grado di risolvere a livello di ADT (cioè a livello di* ***maggiore astrazione e generalità****) molti problemi ignorando i dettagli di questa rappresentazione.*

ADT per Organizzare insiemi di dati

Un modo molto semplice per tenere in memoria tali insiemi consiste nel creare un array per contenerli.

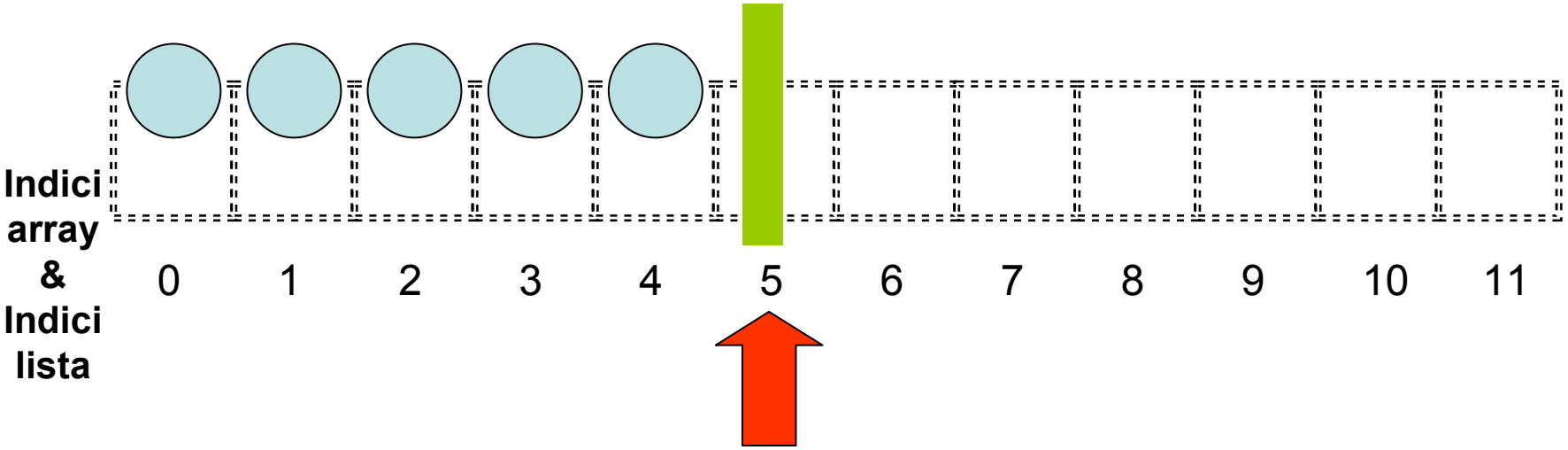


Una prima implementazione con un array che mantiene gli “indirizzi RAM” degli oggetti messi in lista



Implementazione ADT **lista** mediante array: commenti (1)

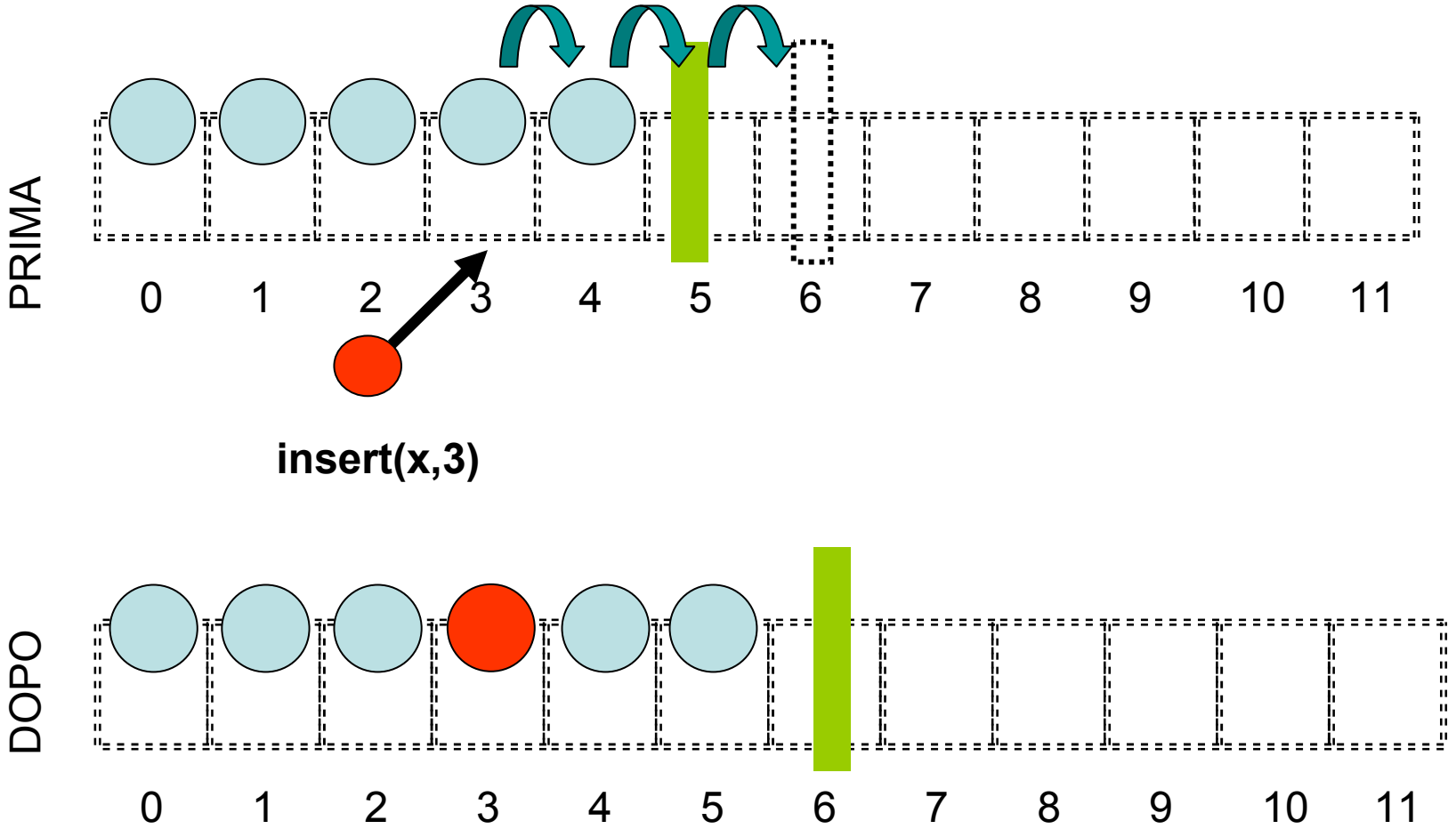
- **END()** : L'array fornisce un "supporto fisso" per sostenere gli elementi della lista. Se l'array viene scelto di dimensione K e la lista in un dato momento contiene solo N<K elementi. Gli ultimi K-N elementi dell'array sono dunque riservati in RAM, ma non sono effettivamente utilizzati dalla lista. Il limite entro il quale ci sono elementi della lista è il numero restituito da questo **metodo** (rappresenta la dimensione corrente dell'array).
- Se si indicizzano gli elementi della lista a partire da 0 questo numero indica anche quanti elementi ci sono nella lista. Nella implementazione con gli array il "confine finale" della lista può per esempio essere mantenuto in una esplicita variabile intera.



In questo caso END() restituisce il valore 5. Nella implementazione con gli array si mantiene una variabile intera endL che in questo caso ha valore 5..

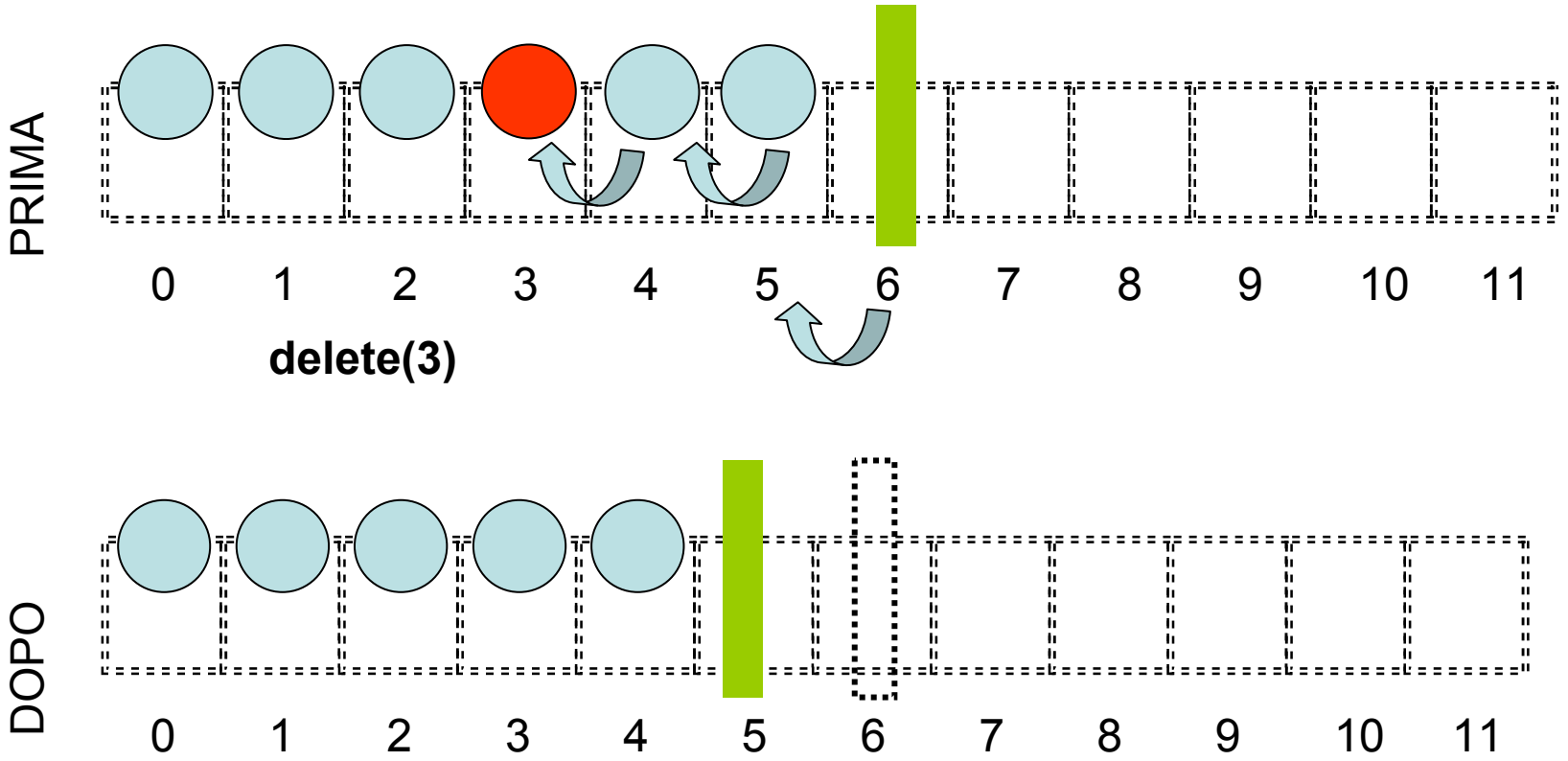
Implementazione ADT **lista** mediante array: commenti (2)

insert(x,p) : *si deve inserire l'elemento x nella posizione p-esima*. Anzitutto si supponga che *p* non ecceda il valore END(). E' chiaro che per fare posto a *x* nel posto *p* gli elementi successivi al posto *p* debbono essere "scivolati" di una posizione e il limite END() va aggiornato. Grazie alla semplice implementazione scelta questo caso prevede l'esecuzione del medesimo codice sia che si inserisca in testa, in coda o nel centro.



Implementazione ADT **lista** mediante array: commenti (3)

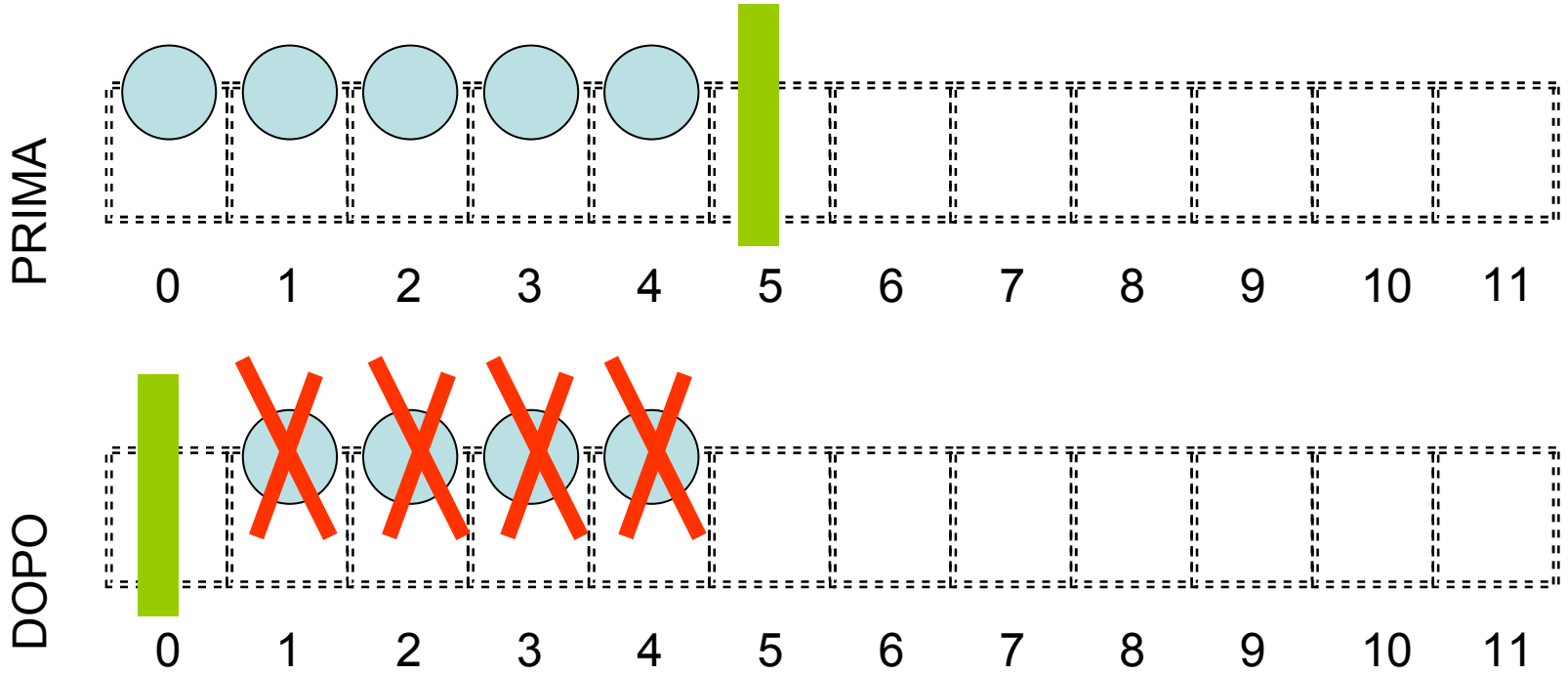
delete(p) : si deve cancellare l'elemento nella posizione p-esima. Per non lasciare un "vuoto" gli elementi successivi al posto *p* debbono essere "scivolati" di una posizione e il limite END() va aggiornato. Grazie alla semplice implementazione scelta questo caso prevede l'esecuzione del medesimo codice sia che si cancelli in testa, in coda o nel centro.



Implementazione ADT **lista** mediante array: commenti (4)

Le operazioni di **locate, retrieve, next, previous** e **stampa** non richiedono particolari accorgimenti e osservazioni. Attenzione al **makeEmpty():** se si muove semplicemente **l'endL** all'inizio dell'array, tutto funziona bene ma… Se non "cancelliamo" gli indirizzi nelle posizioni successive alla prima, tali oggetti saranno ancora referenziabili e la JVM li lascerà in memoria anche se non ci servono più.

E' quindi opportuno cancellare tutti i riferimenti agli oggetti che non vogliamo più utilizzare.



ADT: Liste con Array

Modello: L="sequenza di oggetti"

Operazioni:

END	$L \rightarrow N$
INSERT	$O \times N \times L \rightarrow L$
DELETE	$N \times L \rightarrow L$
LOCATE	$O \times L \rightarrow N$
RETRIEVE	$N \times L \rightarrow O$
STAMPA	$L \rightarrow \Phi$
MAKE_EMPTY	$L \rightarrow L$
Previous, Next	$N \times L \rightarrow O$

ADT per Organizzare insiemi di dati

Gli array presentano però alcuni svantaggi:

- la dimensione deve essere prefissata;
- l'occupazione effettiva di memoria non coincide con le necessità del processo in fase di esecuzione;
- non è possibile allungare un array (se non creandone uno nuovo);
- non è possibile inserire elementi in mezzo (se non spostandone in avanti alcuni).

ADT per Organizzare insiemi di dati

Cosa desideriamo:

- gli elementi devono essere organizzati in un'unica struttura;
- la dimensione del nostro insieme deve poter crescere e decrescere in funzione delle reali necessità del processo in esecuzione.
- deve essere possibile mantenere un ordine nell'insieme;
- deve essere possibile inserire elementi nel mezzo.

Discussione della implementazione delle liste mediante gli array

STUDIARE CON ATTENZIONE IL CODICE `liste01.java`

- **PRO:** semplice, immediato.
- **CONTRO**: alcune operazioni sono realizzate in maniera inefficiente sia per quanto riguarda il numero di passi da compiere sia per quanto riguarda la gestione dello spazio.

- **PROBLEMA 1:** a quanto fissare la dimensione massima della lista? Se la si fissa a 1000 “oggetti” e poi se ne utilizzano meno è evidente lo spreco di memoria. Se invece si fissa a una quota piccola si può rapidamente incappare nel problema dell’esaurimento dello spazio a disposizione.

- **PROBLEMA 2**: alcune operazioni anche semplici (per esempio inserire un oggetto in testa alla lista) richiedono di manipolare ed aggiornare l’intero array: troppo gravoso! Per contro alcune operazioni sono immediate (esempio RETRIEVE).

Dobbiamo/possiamo/vogliamo fare di meglio!!!!