Arrays dinamici

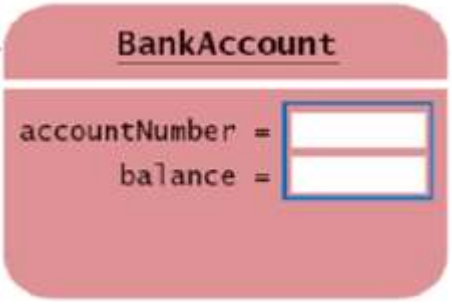
- Un *array dinamico* può crescere e calare di dimensione in base alle necessità

  La classe **ArrayList** (vettore o **lista sequenziale**) gestisce oggetti disposti in sequenza.

  La classe ArrayList fornisce metodi per svolgere le operazioni più comuni, come l'**inserimento** e la **rimozione** di elementi

  La classe ArrayList è una **classe generica**: Array-List<T> contiene oggetti di tipo T.

Se la classe **BankAccount** definisce **oggetti** come illustrato con costruttore a cui passare il numero del c/c

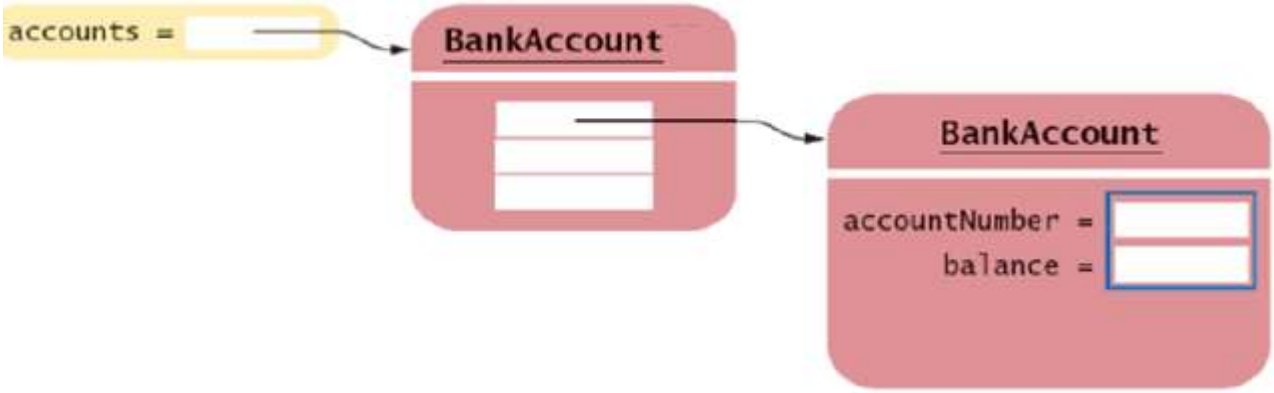



```
public BankAccount (int cc){
        accountNumber = cc;
}
```

Si potrà implementare una ***lista sequenziale*** di tale tipo di oggetti:

ArrayList<BankAccount> accounts **= new ArrayList<BankAccount>();**

accounts**.add(new BankAccount(1001));**
accounts**.add(new BankAccount(1015));**
accounts**.add(new BankAccount(1022));**



Ispezionare gli elementi

Per ispezionare gli oggetti contenuti nel vettore si usa il **metodo get** e **non** l'operatore [ ]

Come con gli array, i valori degli indici iniziano da 0

Ad esempio, accounts.**get** (2) restituisce il conto bancario avente indice 2, cioè il terzo elemento del vettore:

BankAccount anAccount **= accounts.get(2);**
// estrae il terzo elemento della lista con c/c 1022

Accedere a un elemento non esistente è un errore.

L'errore di limiti più frequente è il seguente:

```
int i = accounts.size();
anAccount = accounts.get(i);     // Error
//  valori possibili dell'indice  0. . .i-1
```

NB:

Anche la classe **Vector** come ArrayList (più recente) permette di istanziare **contenitori** di tipi poliformici **"estendibili" e "accorciabili" dinamicamente**.

Prima di JDK5 potevano contenere solo oggetti Object,
in JDK5 (con l'introduzione di classi di tipi generici[1] o ***Generics***) sono ***parametrici rispetto al tipo*** degli oggetti contenuti:

- **Vector** <T> per costruire array estensibile, generico nel tipo T degli elementi (una sequenza di oggetti tipizzati solo in fase di definizione) con metodi sincronizzati (*synchronized*: senza accesso a risorse finché esse sono impegnate da altri metodi)
- **ArrayList** <T> per costruire lista estensibile, generica nel tipo T degli elementi con metodi non sincronizzati (programmazione concorrente).

Aggiungere elementi

 Per **assegnare** un nuovo valore a un elemento esistente di un vettore si usa il **metodo set**:

BankAccount anAccount **= new BankAccount(1729);**
accounts**.set(2,** anAccount**);**

 È possibile **inserire** un oggetto in una posizione intermedia all'interno di un vettore con il **metodo add**.

 L'invocazione accounts.**add** (posizione_i, oggetto) aggiunge l'oggetto nella posizione specificata e sposta tutti gli elementi di una posizione, a partire dall'elemento attualmente in posizione i fino all'ultimo elemento presente nel vettore.

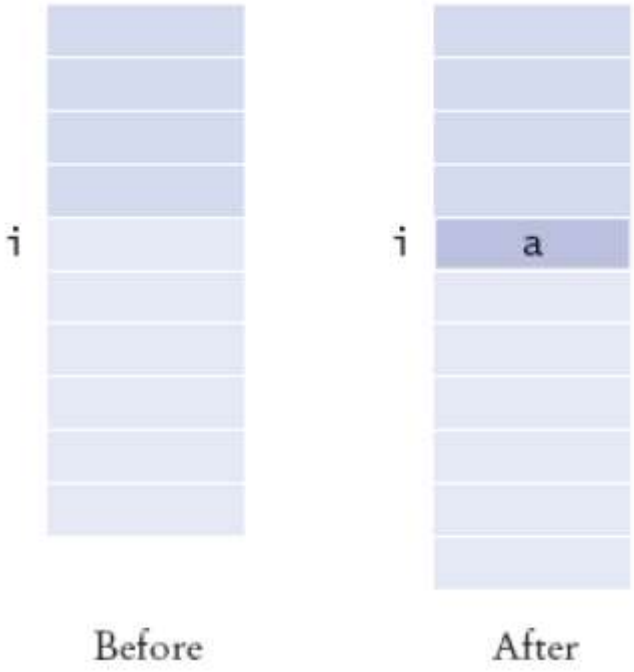


Figura 3
Aggiungere un elemento in una posizione intermedia di un vettore: accounts**.add (i, a)**

allunga di 1 la (parte utilizzata del) l'array

[1] Dopo gli opportuni controlli di consistenza tra tipi, il compilatore **cancella** (Erasure) tutte le informazioni relative ai tipi parametrici, e **sostituisce** (Translation) le relative occorrenze con il tipo di oggetto specifico.

Rimuovere elementi

 Con il metodo **remove** si può **eliminare** l'elemento che si trova in una data posizione, sposta di una posizione all'indietro tutti gli elementi che si trovano dopo l'elemento rimosso e diminuisce di uno la dimensione del vettore.

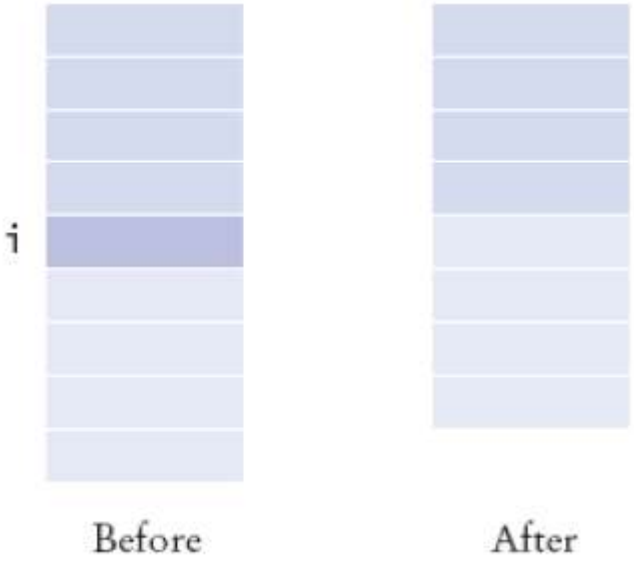


Figura 4
Rimuovere un elemento da una posizione intermedia di un vettore:

accounts.remove (i)

Involucri

 Per poter manipolare valori di tipo primitivo come se fossero oggetti si usano le classi involucro.

```
ArrayList<Double> data = new ArrayList<Double>();
data.add(29.95);   // equivalente a data.add(new Double(29.95));
double x = data.get(0);
```

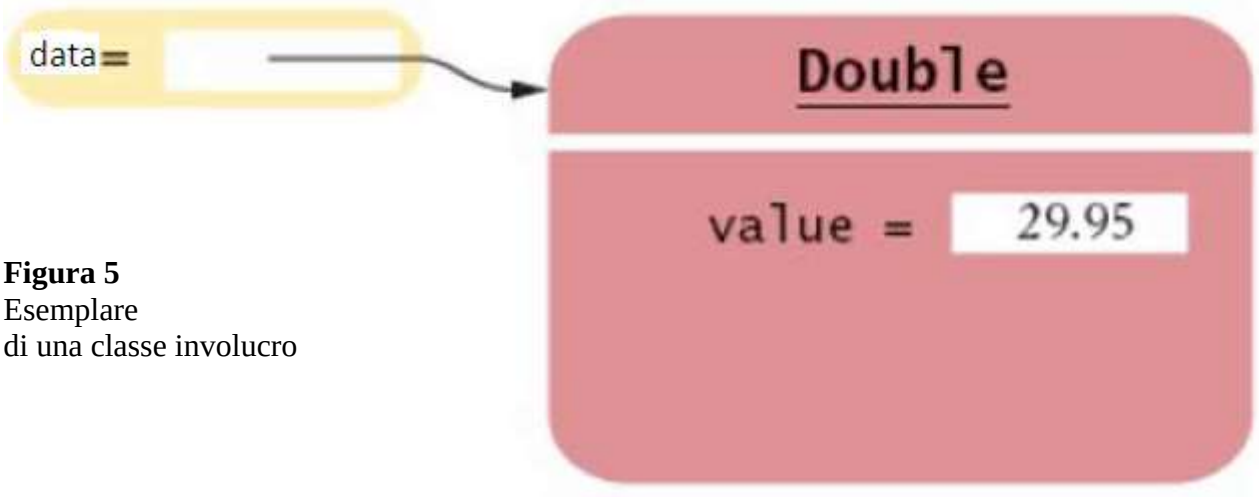


Figura 5
Esemplare di una classe involucro

 Esistono classi involucro per tutti gli otto tipi di dati primitivi

Tipo primitivo	Classe involucro
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Auto-impacchettamento

 A partire dalla versione 5.0 di Java, gli oggetti involucro vengono automaticamente "tolti dalla scatola" per generare valori di tipo primitivo, mediante un processo denominato "**auto-unboxing**"

```
Double d = 29.95;  // auto-boxing
                   // invece di  Double d = new Double(29.95);
                   // da primitivo nell'oggetto wrapper

double x = d; // auto-unboxing
              // invece di  double x = d.doubleValue();
              // da oggetto wrapper in primitivo
```

Auto-boxing

 Le conversioni automatiche funzionano anche all'interno di espressioni aritmetiche (infatti gli operatori possono trattare le classi wrapper come tipi base).

 L'enunciato **Double e = d + 1;**
è valido e significa:

- Converti d in un valore di tipo double
- Aggiungi 1
- Impacchetta il risultato in un nuovo oggetto di tipo Double
- Memorizza in e il riferimento all'oggetto involucro appena creato

 E' lecita la sintassi:

```
Integer i = 5;
i++;          // senza estrazione del campo con metodo
```

Semplici algoritmi per vettori (arrays dinamici)

Contare

 Per contare i valori aventi determinate caratteristiche e presenti in un vettore: ispezionare tutti gli elementi e contare quelli che rispondono ai requisiti, finché non si raggiunge la fine del vettore.

```
public class Bank {

        private ArrayList<BankAccount> accounts;

        /**
                Conta il numero di conti bancari aventi saldo maggiore o uguale al valore indicato.
                @param atLeast il saldo minimo perché un conto venga conteggiato
                @return il numero di conti aventi saldo maggiore o uguale al valore indicato.
        */
        public int count (double atLeast)  {

                int matches = 0;
                for (int i = 1; i < accounts.size(); i++) {
                        BankAccount a = accounts.get(i);
                        if (a.getBalance() >= atLeast)
                           matches++;   // trovato
                }
                return matches;
        }
        . . .
}
```

Trovare un valore

 Per trovare un valore in un vettore occorre controllarne tutti gli elementi finché non si trova il valore cercato.

```
public class Bank {

        /**
                Verifica se la banca contiene un conto con il numero indicato.
                @param accountNumber il numero di conto da cercare
                @return il conto con il numero indicato, oppure null se tale conto non esiste
        */
        public BankAccount  find (int accountNumber) {

                for (int i = 1; i < accounts.size(); i++) {
                        BankAccount a = accounts.get(i);
                        if (a.getAccountNumber() == accountNumber)  // trovato
                          return a;
                }
                return null; // nessuna occorrenza in tutta l'array list
        }
        . . .
}
```

Trovare il massimo (o il minimo)

 Questo metodo funziona soltanto se il vettore contiene **almeno un elemento**: non ha senso cercare l'elemento di valore maggiore in un insieme vuoto.

 Se l'insieme è vuoto, restituisce null

```
if (accounts.size() == 0)
   return null;
BankAccount largestYet = accounts.get(0);
. . .
```

```
public class Bank  {

      /**
            Restituisce il conto bancario avente il saldo maggiore.
            @return il conto con il saldo maggiore, oppure null se la banca non ha conti
      */
      public BankAccount getMaximum()  {

            if (accounts.size() == 0)
               return null;
            BankAccount largestYet = accounts.get(0);

            for (int i = 1; i < accounts.size(); i++)  {

                  BankAccount a = accounts.get(i);
                   if (a.getBalance() > largestYet.getBalance())
                         largestYet = a;
            }
            return largestYet;
      }
      . . .
}
```

Ispezionare[2] gli elementi e calcolare il saldo totale

 Integrare, nella definizione della classe **Bank**, tale metodo noto che il ciclo seguente calcola il ***saldo totale*** di tutti i conti bancari:

```
double sum = 0;
for (int i = 0; i < accounts.size(); i++) {
  BankAccount a = accounts.get(i);
  sum = sum + a.getBalance();
}
```

[2] Per ***scandire*** gli elementi di una ***collezione***: for generalizzato (migliorato) e forEach dalla versione Java 8.