Questi lucidi provengono dal Capitolo 7 del libro

Cay S. Horstmann
Concetti di informatica e fondamenti di Java
quarta edizione

Vettori

- La classe `ArrayList` (vettore o lista sequenziale) gestisce oggetti disposti in sequenza.
- Un vettore può crescere e calare di dimensione in base alle necessità
- 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: `ArrayList<T>` contiene oggetti di tipo `T`.

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
accounts.add(new BankAccount(1001));
accounts.add(new BankAccount(1015));
accounts.add(new BankAccount(1022));
```

- Il metodo `size` restituisce la dimensione attuale del vettore

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);
   // fornisce il terzo elemento del vettore
```

- 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); // Errore
// gli indici validi vanno da 0 a i-1
```

Aggiungere elementi

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

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

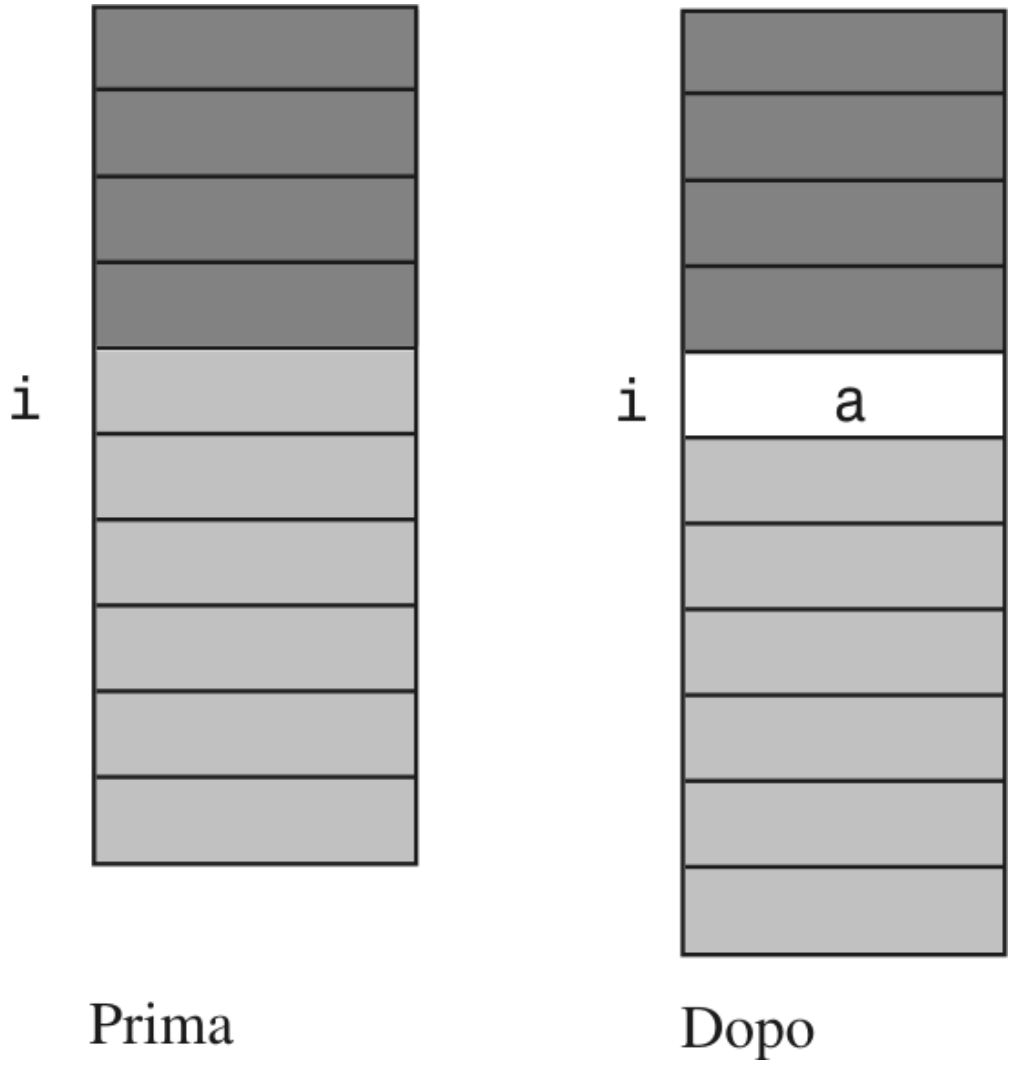
- È possibile inserire un oggetto in una posizione intermedia all'interno di un vettore.

```
accounts.add(i, a)
```

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

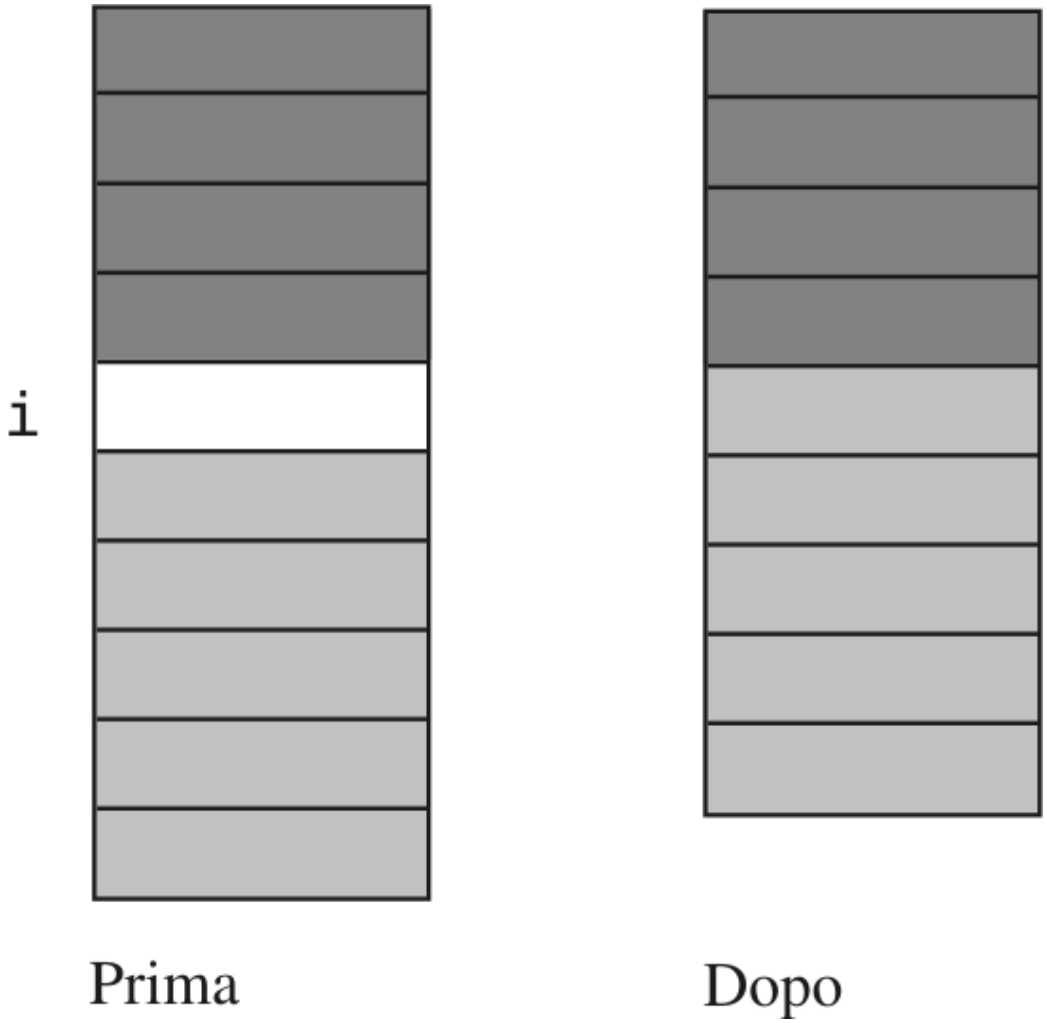
Aggiungere elementi

Figura 3
Aggiungere un elemento in una posizione intermedia di un vettore



Rimuovere elementi

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



L'invocazione `accounts.remove(i)` elimina l'elemento che si trova in posizione i, sposta di una posizione all'indietro tutti gli elementi che si trovano dopo l'elemento rimosso e diminuisce di uno la dimensione del vettore.

File: ArrayListTester.java

```
01: import java.util.ArrayList;
02:
03: /**
04:    Questo programma collauda la classe ArrayList.
05: */
06: public class ArrayListTester
07: {
08:    public static void main(String[] args)
09:    {
10:       ArrayList<BankAccount> accounts
11:             = new ArrayList<BankAccount>();
12:       accounts.add(new BankAccount(1001));
13:       accounts.add(new BankAccount(1015));
14:       accounts.add(new BankAccount(1729));
15:       accounts.add(1, new BankAccount(1008));
16:       accounts.remove(0);
```

Segue

File: ArrayListTester.java

```
17:
18:       System.out.println("size=" + accounts.size());
19:       BankAccount first = accounts.get(0);
20:       System.out.println("first account number="
21:             + first.getAccountNumber());
22:       BankAccount last = accounts.get(accounts.size() - 1);
23:       System.out.println("last account number="
24:             + last.getAccountNumber());
25:    }
26: }
```

File: BankAccount.java

```
01: /**
02:    Un conto bancario ha un saldo che può essere modificato
03:    da depositi e prelievi.
04: */
05: public class BankAccount
06: {
07:    /**
08:       Costruisce un conto bancario con saldo uguale a zero.
09:       @param anAccountNumber il numero di questo conto bancario
10:    */
11:    public BankAccount(int anAccountNumber)
12:    {
13:       accountNumber = anAccountNumber;
14:       balance = 0;
15:    }
16:
```

Segue

File: BankAccount.java

```
17:    /**
18:       Costruisce un conto bancario con un saldo assegnato.
19:       @param anAccountNumber il numero di questo conto bancario
20:       @param initialBalance il saldo iniziale
21:    */
22:    public BankAccount(int anAccountNumber, double initialBalance)
23:    {
24:       accountNumber = anAccountNumber;
25:       balance = initialBalance;
26:    }
27:
28:    /**
29:       Restituisce il numero di conto del conto bancario.
30:       @return il numero di conto
31:    */
32:    public int getAccountNumber()
33:    {
34:       return accountNumber;
35:    }
```

Segue

File: BankAccount.java

```
36:
37:    /**
38:       Versa denaro nel conto bancario.
39:       @param amount l'importo da versare
40:    */
41:    public void deposit(double amount)
42:    {
43:       double newBalance = balance + amount;
44:       balance = newBalance;
45:    }
46:
47:    /**
48:       Preleva denaro dal conto bancario.
49:       @param amount l'importo da prelevare
50:    */
51:    public void withdraw(double amount)
52:    {
53:       double newBalance = balance - amount;
54:       balance = newBalance;
```

Segue

File: BankAccount.java

```
55:    }
56:
57:    /**
58:       Ispeziona il valore del saldo attuale del conto bancario.
59:       @return il saldo attuale
60:    */
61:    public double getBalance()
62:    {
63:       return balance;
64:    }
65:
66:    private int accountNumber;
67:    private double balance;
68: }
```

File: `BankAccount.java`

Output:

```
Size: 3

First account number: 1008

Last account number: 1729
```

Il ciclo `for` generalizzato

- Il ciclo `for` generalizzato scandisce tutti gli elementi di una raccolta:

```
double[] data = . . .;
double sum = 0;
for (double e : data) // si legge "per ogni e in data"
{
   sum = sum + e;
}
```

Segue

Il ciclo `for` generalizzato

- Per scandire tutti gli elementi di un array non è obbligatorio utilizzare il ciclo `for` generalizzato: lo stesso ciclo può essere realizzato con un `for` normale e una variabile indice esplicita.

```
double[] data = . . .;
double sum = 0;
for (int i = 0; i < data.length; i++)
{
   double e = data[i];
   sum = sum + e;
}
```

Il ciclo `for` generalizzato

- Il ciclo `for` generalizzato può essere usato anche per ispezionare tutti gli elementi di un vettore.
  Ad esempio, il ciclo seguente calcola il saldo totale di tutti i conti bancari:

```
ArrayList<BankAccount> accounts = . . . ;
double sum = 0;
for (BankAccount a : accounts)
{
   sum = sum + a.getBalance();
}
```

- Il ciclo è equivalente a questo ciclo “normale”:

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

Sintassi 8.3: Il ciclo `for` generalizzato

```
for (Tipo variabile : aggregato)
    istruzioneInterna
```

Esempio:

```
for (double e : data)
    sum = sum + e;
```

Obiettivo:

Eseguire un ciclo avente un'iterazione per ogni elemento appartenente a un aggregato. All'inizio di ciascuna iterazione viene assegnato alla variabile l'elemento successivo dell'aggregato, poi viene eseguita l'istruzioneInterna.