



Capitolo 5

Array e collezioni

Sommario:

Array di oggetti

Dimensione di un array

Accesso agli elementi di un array

Array e cicli `for` e *for-each*

Inizializzazione di array

L'intestazione del metodo `main`

Array di tipo primitivo

Array di array

Sottoproblemi e sottoprogrammi

Classi generiche

La classe generica `Sequenza<E>`

La classe generica `SequenzaOrdinata<E>`

Array

Insieme ordinato di variabili dello *stesso tipo* (*tipo base*), ognuna delle quali è accessibile specificando la posizione in cui si trova.

Array

Insieme ordinato di variabili dello *stesso tipo* (*tipo base*), ognuna delle quali è accessibile specificando la posizione in cui si trova.

- *Tipo base*

Può essere sia un tipo primitivo sia un tipo riferimento

Array

Insieme ordinato di variabili dello *stesso tipo* (*tipo base*), ognuna delle quali è accessibile specificando la posizione in cui si trova.

- ▶ **Tipo base**

Può essere sia un tipo primitivo sia un tipo riferimento

- ▶ **Array di oggetti**

Array il cui tipo base è un tipo riferimento

- ▶ In Java gli array sono **oggetti**

Costruzione di array

- In Java gli array sono **oggetti**

Costruzione di un array

```
new tipo_base[espressione_int]
```

Costruzione di array

- In Java gli array sono **oggetti**

Costruzione di un array

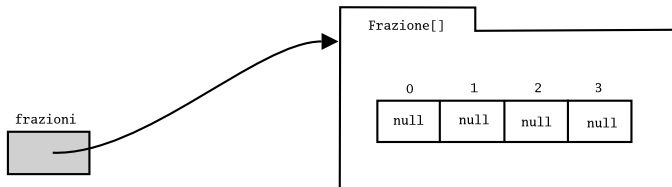
```
new tipo_base[espressione_int]
```

Dichiarazione di variabile

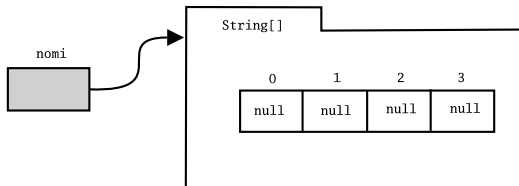
```
Tipo_base[] identificatore;
```



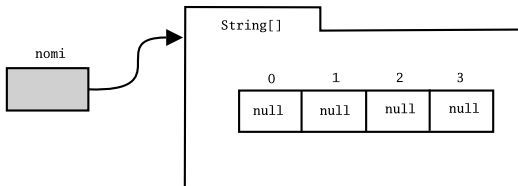
```
Frazione[] frazioni;  
frazioni = new Frazione[4];
```



```
String[] nomi = new String[4]
```



```
String[] nomi = new String[4]
```



- In un array di oggetti le posizioni sono **automaticamente inizializzate** a **null** all'atto della creazione

- ▶ Ogni oggetto di tipo array ha memorizzata nel suo stato l'informazione relativa alla sua *dimensione*

Dimensione di un array

- ▶ Ogni oggetto di tipo array ha memorizzata nel suo stato l'informazione relativa alla sua *dimensione*
- ▶ Tale informazione si trova in un *campo* di nome `length` e di tipo `int`

- ▶ Ogni oggetto di tipo array ha memorizzata nel suo stato l'informazione relativa alla sua *dimensione*
- ▶ Tale informazione si trova in un *campo* di nome *length* e di tipo *int*

Esempio

```
Frazione[] frazioni;  
frazioni = new Frazione[4];
```

Dimensione di un array

- ▶ Ogni oggetto di tipo array ha memorizzata nel suo stato l'informazione relativa alla sua *dimensione*
- ▶ Tale informazione si trova in un *campo* di nome `length` e di tipo `int`

Esempio

```
Frazione[] frazioni;  
frazioni = new Frazione[4];
```

```
frazioni.length
```

- ▶ È un'espressione di tipo `int`
- ▶ Il suo valore è `4`

Accesso agli elementi di un array

Accesso agli elementi

nome_array[*selettore*]

Accesso agli elementi di un array

Accesso agli elementi

nome_array[*selettore*]

► *selettore*

Dev'essere un'**espressione** di tipo **int**

Espressioni di tipo **short**, **byte**, o **char** vengono promosse automaticamente a **int**

Accesso agli elementi

nome_array[*selettore*]

- ▶ *selettore*

Dev'essere un'espressione di tipo **int**

Espressioni di tipo **short**, **byte**, o **char** vengono promosse automaticamente a **int**

- ▶ Le posizioni di un array sono contate a partire da **zero**

Accesso agli elementi di un array

Accesso agli elementi

nome_array[*selettore*]

- ▶ *selettore*

Dev'essere un'espressione di tipo **int**

Espressioni di tipo **short**, **byte**, o **char** vengono promosse automaticamente a **int**

- ▶ Le posizioni di un array sono contate a partire da **zero**

- ▶ *nome_array*[*selettore*] è una variabile con:
 tipo: tipo base dell'array

Accesso agli elementi di un array

Accesso agli elementi

nome_array[*selettore*]

- ▶ *selettore*

Dev'essere un'espressione di tipo **int**

Espressioni di tipo **short**, **byte**, o **char** vengono promosse automaticamente a **int**

- ▶ Le posizioni di un array sono contate a partire da **zero**

- ▶ *nome_array*[*selettore*] è una variabile con:

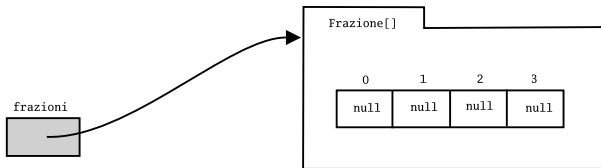
tipo: tipo base dell'array

valore: il contenuto della posizione corrispondente dell'array

```
Frazione[] frazioni;  
frazioni = new Frazione[4];  
  
frazioni[0] = new Frazione(1,4);  
frazioni[1] = new Frazione(2,4);  
frazioni[2] = new Frazione(3,4);  
  
int i = 2;  
frazioni[2 * i - 1] = frazioni[2 * i - 2].piu(frazioni[1]);
```

Accesso alle componenti

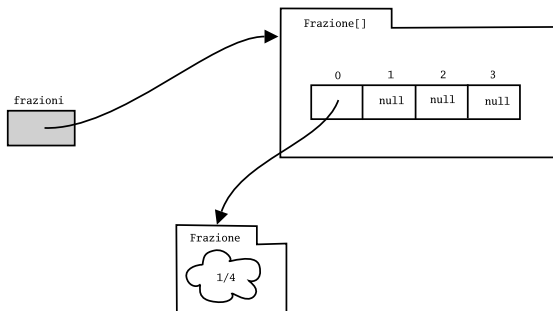
```
Frazione[] frazioni = new Frazione[4];
```



Accesso alle componenti

```
Frazione[] frazioni = new Frazione[4];
```

```
frazioni[0] = new Frazione(1,4);
```



Accesso alle componenti

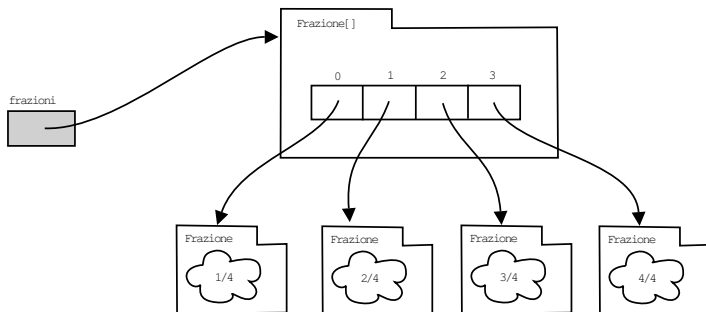
```
Frazione[] frazioni = new Frazione[4];
```

```
frazioni[0] = new Frazione(1,4);
```

```
frazioni[1] = new Frazione(2,4);
```

```
frazioni[2] = new Frazione(3,4);
```

```
frazioni[3] = new Frazione(4,4);
```



- ▶ Il tentativo di accedere a una componente non definita dell'array causa un **errore in fase di esecuzione**

- Il tentativo di accedere a una componente non definita dell'array causa un **errore in fase di esecuzione**

```
Frazione[] frazioni = new Frazione[4];  
...  
frazioni[4] = new Frazione(5,4);
```

- Il tentativo di accedere a una componente non definita dell'array causa un **errore in fase di esecuzione**

```
Frazione[] frazioni = new Frazione[4];  
...  
frazioni[4] = new Frazione(5,4);
```

```
Exception in thread "main"  
    java.lang.ArrayIndexOutOfBoundsException: 4  
    ...
```

```
Frazione[] frazioni = new Frazione[4];  
  
for (int i = 0; i < frazioni.length; i++)  
    frazioni[i] = new Frazione(i + 1,4);  
  
for (int i = 0; i < frazioni.length; i++)  
    out.println(frazioni[i].toString());
```

Sintassi

```
for (tipo_base identificatore: array)  
    istruzione
```

Sintassi

```
for (tipo_base identificatore: array)  
    istruzione
```

Esempio

```
Frazione[] frazioni = new Frazione[4];  
...  
for (Frazione f: frazioni)  
    out.println(f.toString());
```

Sintassi

```
for (tipo_base identificatore: array)  
    istruzione
```

Esempio

```
Frazione[] frazioni = new Frazione[4];  
...  
for (Frazione f: frazioni)  
    out.println(f.toString());
```

è equivalente a:

```
for (int i = 0; i < frazioni.length; i++)  
    out.println(frazioni[i].toString());
```

- ▶ Consente di ottenere uno dopo l'altro i valori contenuti nell'array

Osservazione: *for-each*

- ▶ Consente di ottenere uno dopo l'altro i valori contenuti nell'array
- ▶ Non consente di accedere alle posizioni dell'array e quindi non consente di modificare l'array

- ▶ Consente di ottenere uno dopo l'altro i valori contenuti nell'array
- ▶ Non consente di accedere alle posizioni dell'array e quindi non consente di modificare l'array

Esempio

```
Frazione[] frazioni = new Frazione[4];  
  
for (int i = 0; i < frazioni.length; i++)  
    frazioni[i] = new Frazione(i + 1,4);
```

- ▶ Consente di ottenere uno dopo l'altro i valori contenuti nell'array
- ▶ Non consente di accedere alle posizioni dell'array e quindi non consente di modificare l'array

Esempio

```
Frazione[] frazioni = new Frazione[4];  
  
for (int i = 0; i < frazioni.length; i++)  
    frazioni[i] = new Frazione(i + 1,4);  
  
for (Frazione f: frazioni)  
    out.println(f.toString());
```

Inizializzazione di array

- Specificando fra parentesi graffe la sequenza di valori che costituiscono gli elementi dell'array

Inizializzazione di array

- Specificando fra parentesi graffe la sequenza di valori che costituiscono gli elementi dell'array

```
Frazione[] frazioni = {new Frazione(1,4), new Frazione(2,4),  
                        new Frazione(3,4), new Frazione(4,4)};
```

Inizializzazione di array

- Specificando fra parentesi graffe la sequenza di valori che costituiscono gli elementi dell'array

```
Frazione[] frazioni = {new Frazione(1,4), new Frazione(2,4),  
                        new Frazione(3,4), new Frazione(4,4)};
```

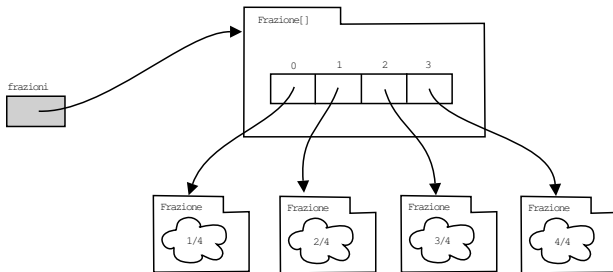
- La dimensione dell'array viene **dedotta dal compilatore**

Inizializzazione di array

- Specificando fra parentesi graffe la sequenza di valori che costituiscono gli elementi dell'array

```
Frazione[] frazioni = {new Frazione(1,4), new Frazione(2,4),  
                        new Frazione(3,4), new Frazione(4,4)};
```

- La dimensione dell'array viene **dedotta dal compilatore**



Esempio: SequenzaFrazioni

```
...  
final int MAX = 10;  
Frazione[] frazioni = new Frazione[MAX];
```


Esempio: SequenzaFrazioni

```
...
final int MAX = 10;
Frazione[] frazioni = new Frazione[MAX];

//fase lettura
for (int pos = 0; pos < frazioni.length; pos++) {
    out.println("Lettura della frazione " + (pos + 1));
    int num = in.readInt("Numeratore: ");
    int den = in.readInt("Denominatore: ");
    frazioni[pos] = new Frazione(num,den);
}
```

Esempio: SequenzaFrazioni

```
...
final int MAX = 10;
Frazione[] frazioni = new Frazione[MAX];

//fase lettura
for (int pos = 0; pos < frazioni.length; pos++) {
    out.println("Lettura della frazione " + (pos + 1));
    int num = in.readInt("Numeratore: ");
    int den = in.readInt("Denominatore: ");
    frazioni[pos] = new Frazione(num,den);
}

//fase di scrittura
for (Frazione f: frazioni)
    out.println(f.toString());
```

```
String[] nomi;  
  
//fase di scrittura  
for (int pos = 0; pos < nomi.length; pos++)  
    out.println(nomi[pos].toString());
```

Errore: for

```
String[] nomi;  
  
//fase di scrittura  
for (int pos = 0; pos < nomi.length; pos++)  
    out.println(nomi[pos].toString());
```

Compilazione

```
> javac UsoErratoArray.java  
...: variable nomi might not have been initialized  
    for (int pos = 0; pos < nomi.length; pos++)  
        ^  
1 error
```

Errore: *for-each*

```
String[] nomi;  
  
//fase di scrittura  
for (String s: nomi)  
    out.println(s.toString());
```

Errore: *for-each*

```
String[] nomi;  
  
//fase di scrittura  
for (String s: nomi)  
    out.println(s.toString());
```

Compilazione

```
> javac UsoErratoArray.java  
...: variable nomi might not have been initialized  
    for (String s: nomi)  
        ^  
1 error
```

```
Frazione[] frazioni = new Frazione[4];  
  
//fase di scrittura  
for (int pos = 0; pos < frazioni.length; pos++)  
    out.println(frazioni[pos].toString());
```

```
Frazione[] frazioni = new Frazione[4];  
  
//fase di scrittura  
for (int pos = 0; pos < frazioni.length; pos++)  
    out.println(frazioni[pos].toString());
```

Esecuzione

```
> java UsoErratoArray  
Exception in thread "main" java.lang.NullPointerException  
    at UsoErratoArray.main(UsoErratoArray.java:14)
```


Errore: *for-each*

```
Frazione[] frazioni = new Frazione[4];  
  
//fase di scrittura  
for (Frazione f: frazioni)  
    out.println(f.toString());
```

```
Frazione[] frazioni = new Frazione[4];  
  
//fase di scrittura  
for (Frazione f: frazioni)  
    out.println(f.toString());
```

Esecuzione

```
> java UsoErratoArray  
Exception in thread "main" java.lang.NullPointerException  
    at UsoErratoArray.main(UsoErratoArray.java:14)
```

L'intestazione del metodo main

```
public static void main(String[] args)
```

L'intestazione del metodo main

```
public static void main(String[] args)
```

► `static`

È un metodo statico

```
public static void main(String[] args)
```

- ▶ `static`

È un metodo statico

- ▶ `void`

Non restituisce alcun valore

```
public static void main(String[] args)
```

- ▶ `static`

È un metodo statico

- ▶ `void`

Non restituisce alcun valore

- ▶ `String[] args`

Riceve come argomento il riferimento ad un array di stringhe

```
public static void main(String[] args)
```

Quando si manda in esecuzione una classe:

- ▶ La JVM cerca nel *bytecode della classe* un metodo statico con il prototipo descritto sopra

Esecuzione del metodo main

```
public static void main(String[] args)
```

Quando si manda in esecuzione una classe:

- ▶ La JVM cerca nel *bytecode della classe* un metodo statico con il prototipo descritto sopra
- ▶ Se lo trova, lo invoca passandogli come argomento il riferimento all'*array contenente le stringhe specificate nel comando di esecuzione* dopo il nome della classe

Esecuzione del metodo main

```
public static void main(String[] args)
```

Quando si manda in esecuzione una classe:

- ▶ La JVM cerca nel *bytecode della classe* un metodo statico con il prototipo descritto sopra
- ▶ Se lo trova, lo invoca passandogli come argomento il riferimento all'*array contenente le stringhe specificate nel comando di esecuzione* dopo il nome della classe
- ▶ Se non vengono forniti argomenti è un riferimento all'array vuoto (cioè *args.length* vale 0)

```
import prog.io.*;

class Ripeti {
    public static void main(String[] args) {
        ConsoleOutputManager out = new ConsoleOutputManager();

        for (String s : args)
            out.println(s);
    }
}
```

Esempio

```
import prog.io.*;

class Ripeti {
    public static void main(String[] args) {
        ConsoleOutputManager out = new ConsoleOutputManager();

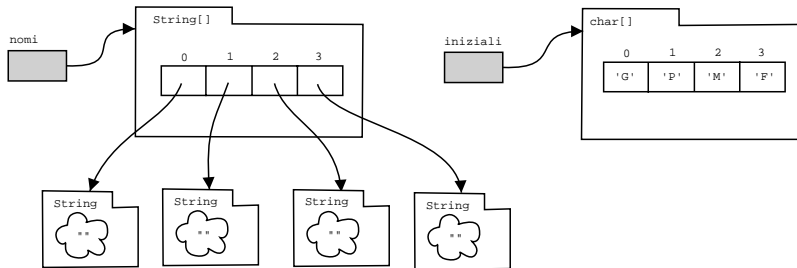
        for (String s : args)
            out.println(s);
    }
}
```

Esecuzione

```
> java Ripeti PIPPO pluto
PIPP0
pluto
```

Differenza fra array di oggetti e di tipo primitivo

```
String[] nomi = {"", "", "", ""};  
char[] iniziali = {'G', 'P', 'M', 'F'};
```

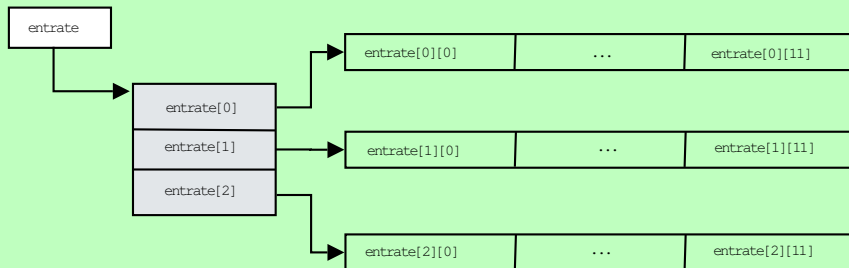


Array multidimensionali

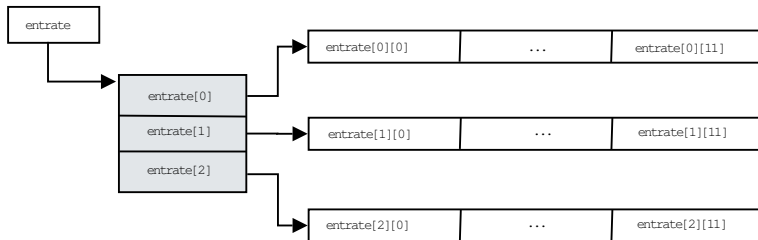
Array i cui elementi sono a loro volta array

Array bidimensionale (matrice)

```
int ANNI = 3;  
int MESI = 12;  
Importo[] [] entrate = new Importo[ANNI][MESI];
```



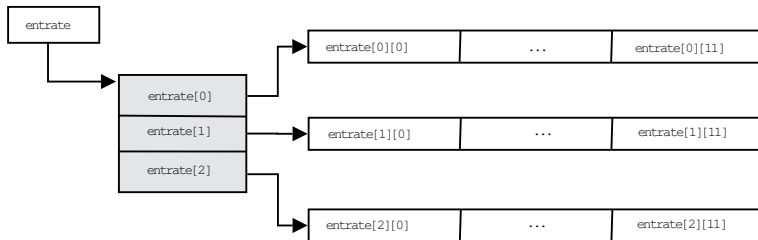
Dichiarazione e accesso



- **Accesso** all'elemento memorizzato nella **seconda riga** e **terza colonna**:

```
Importo i = entrate[1][2];
```

Dichiarazione e accesso



- ▶ **Accesso** all'elemento memorizzato nella **seconda riga** e **terza colonna**:

```
Importo i = entrate[1][2];
```

- ▶ **Dimensioni**:
 - ▶ Il numero delle righe è `entrate.length`
 - ▶ Il numero delle colonne della riga `i` è `entrate[i].length`

Sottoproblemi e sottoprogrammi

- Un meccanismo indispensabile per gestire problemi complessi è la *scomposizione in sottoproblemi*.

Sottoproblemi e sottoprogrammi

- ▶ Un meccanismo indispensabile per gestire problemi complessi è la *scomposizione in sottoproblemi*.
- ▶ Il suo corrispettivo nella programmazione consiste nella *suddivisione di un programma in sottoprogrammi*.

Sottoproblemi e sottoprogrammi

- ▶ Un meccanismo indispensabile per gestire problemi complessi è la *scomposizione in sottoproblemi*.
- ▶ Il suo corrispettivo nella programmazione consiste nella *suddivisione di un programma in sottoprogrammi*.

Esempio

- ▶ Si scriva un'applicazione che letti due vettori di numeri interi della stessa lunghezza, calcoli il vettore somma e visualizzi sul monitor i due vettori letti e il vettore somma.

Sottoproblemi e sottoprogrammi

- ▶ Un meccanismo indispensabile per gestire problemi complessi è la *scomposizione in sottoproblemi*.
- ▶ Il suo corrispettivo nella programmazione consiste nella *suddivisione di un programma in sottoprogrammi*.

Esempio

- ▶ Si scriva un'applicazione che letti due vettori di numeri interi della stessa lunghezza, calcoli il vettore somma e visualizzi sul monitor i due vettori letti e il vettore somma.
- ▶ Possiamo rappresentare lo schema del programma riducendolo ad una sequenza di sottoproblemi che già sappiamo risolvere

```
class SommaVettori {  
    public static void main(String[] args) {  
        leggi la lunghezza dei vettori  
        leggi il primo vettore  
        leggi il secondo vettore  
        calcola il vettore somma  
        costruisci le stringhe che rappresentano i tre vettori  
        scrivi le stringhe  
    }  
}
```

SommaVettori (1)

```
...  
//leggi la lunghezza dei vettori  
int lunghezza = in.readInt("Lunghezza dei vettori? ");
```

SommaVettori (1)

```
...
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = new int[lunghezza];
for (int i = 0; i < vett1.length; i++)
    vett1[i] = in.readInt("Elemento " + i + "? ");
```

SommaVettori (1)

```
...
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = new int[lunghezza];
for (int i = 0; i < vett1.length; i++)
    vett1[i] = in.readInt("Elemento " + i + "? ");

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = new int[lunghezza];
for (int i = 0; i < vett2.length; i++)
    vett2[i] = in.readInt("Elemento " + i + "? ");
...
```

SommaVettori (1)

```
...
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = new int[lunghezza];
for (int i = 0; i < vett1.length; i++)
    vett1[i] = in.readInt("Elemento " + i + "? ");

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = new int[lunghezza];
for (int i = 0; i < vett2.length; i++)
    vett2[i] = in.readInt("Elemento " + i + "? ");
...
```

Osservazione

- ▶ Le istruzioni per effettuare la lettura del primo e del secondo vettore si differenziano solo per il nome della variabile utilizzata, nel primo caso **vett1**, nel secondo **vett2**

SommaVettori (1)

```
...
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = new int[lunghezza];
for (int i = 0; i < vett1.length; i++)
    vett1[i] = in.readInt("Elemento " + i + "? ");

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = new int[lunghezza];
for (int i = 0; i < vett2.length; i++)
    vett2[i] = in.readInt("Elemento " + i + "? ");
...
```

Osservazione

- ▶ Le istruzioni per effettuare la lettura del primo e del secondo vettore si differenziano solo per il nome della variabile utilizzata, nel primo caso **vett1**, nel secondo **vett2**
- ▶ I due blocchi risolvono lo stesso problema ma operano su dati differenti.

SommaVettori (2)

```
//calcola il vettore somma  
int[] somma = new int[lunghezza];  
for (int i = 0; i < somma.length; i++)  
    somma[i] = vett1[i] + vett2[i];
```

SommaVettori (2)

```
//calcola il vettore somma
int[] somma = new int[lunghezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];

//costruisci le stringhe che rappresentano i tre vettori
//strVett1, strVett2 e strSomma
...
```

SommaVettori (2)

```
//calcola il vettore somma
int[] somma = new int[lunghezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];

//costruisci le stringhe che rappresentano i tre vettori
//strVett1, strVett2 e strSomma
...

//scrivi le stringhe
out.println("Vettore 1: [" + strVett1 + "]");
out.println("Vettore 2: [" + strVett2 + "]");
out.println("Vettore 3: [" + strSomma + "]");
```

SommaVettori (3): costruzione delle stringhe

```
...  
//costruisci le stringhe che rappresentano i tre vettori  
String strVett1 = "";  
for (int i = 0; i < vett1.length; i++)  
    strVett1 += vett1[i] + (i < vett1.length - 1 ? " " : "");
```

SommaVettori (3): costruzione delle stringhe

```
...
//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = "";
for (int i = 0; i < vett1.length; i++)
    strVett1 += vett1[i] + (i < vett1.length - 1 ? " " : "");

String strVett2 = "";
for (int i = 0; i < vett2.length; i++)
    strVett2 += vett2[i] + (i < vett2.length - 1 ? " " : "");
```

SommaVettori (3): costruzione delle stringhe

```
...
//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = "";
for (int i = 0; i < vett1.length; i++)
    strVett1 += vett1[i] + (i < vett1.length - 1 ? " " : "");

String strVett2 = "";
for (int i = 0; i < vett2.length; i++)
    strVett2 += vett2[i] + (i < vett2.length - 1 ? " " : "");

String strSomma= "";
for (int i = 0; i < somma.length; i++)
    strSomma += somma[i] + (i < somma.length - 1 ? " " : "");
...
```

SommaVettori (3): costruzione delle stringhe

```
...
//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = "";
for (int i = 0; i < vett1.length; i++)
    strVett1 += vett1[i] + (i < vett1.length - 1 ? " " : "");

String strVett2 = "";
for (int i = 0; i < vett2.length; i++)
    strVett2 += vett2[i] + (i < vett2.length - 1 ? " " : "");

String strSomma= "";
for (int i = 0; i < somma.length; i++)
    strSomma += somma[i] + (i < somma.length - 1 ? " " : "");
...
```

SommaVettori (3): costruzione delle stringhe

```
...
//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = "";
for (int i = 0; i < vett1.length; i++)
    strVett1 += vett1[i] + (i < vett1.length - 1 ? " " : "");

String strVett2 = "";
for (int i = 0; i < vett2.length; i++)
    strVett2 += vett2[i] + (i < vett2.length - 1 ? " " : "");

String strSomma= "";
for (int i = 0; i < somma.length; i++)
    strSomma += somma[i] + (i < somma.length - 1 ? " " : "");
...
```

Osservazione

- ▶ Le istruzioni per costruire le stringhe si differenziano unicamente per il nome della stringa generata e per il nome dell'array utilizzato.
- ▶ I tre blocchi risolvono lo stesso problema ma operano su dati differenti.

- Supponiamo di disporre di una classe `C` con due metodi che realizzino la soluzione dei sottoproblemi che abbiamo evidenziato.

- ▶ Supponiamo di disporre di una classe `C` con due metodi che realizzino la soluzione dei sottoproblemi che abbiamo evidenziato.
- ▶ `static int[] leggiVettore(ConsoleInputManager input,
int lung)`

Restituisce il riferimento a un array di `int` della lunghezza specificata tramite il secondo parametro. L'array è inizializzato con valori letti dal canale riferito da `input`.

- ▶ Supponiamo di disporre di una classe `C` con due metodi che realizzino la soluzione dei sottoproblemi che abbiamo evidenziato.

- ▶ `static int[] leggiVettore(ConsoleInputManager input,
int lung)`

Restituisce il riferimento a un array di `int` della lunghezza specificata tramite il secondo parametro. L'array è inizializzato con valori letti dal canale riferito da `input`.

- ▶ `static String generaStringa(int[] vettore)`

Restituisce il riferimento a una stringa che rappresenta il contenuto dell'array di interi di cui viene fornito il riferimento come argomento.

SommaVettori (5): nuova soluzione

```
//leggi la lunghezza dei vettori  
int lunghezza = in.readInt("Lunghezza dei vettori? ");
```

SommaVettori (5): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = C.leggiVettore(in, lunghezza);
```

SommaVettori (5): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = C.leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = C.leggiVettore(in, lunghezza);
```

SommaVettori (5): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = C.leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = C.leggiVettore(in, lunghezza);

//calcola il vettore somma
int[] somma = new int[lughezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];
```

SommaVettori (5): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = C.leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = C.leggiVettore(in, lunghezza);

//calcola il vettore somma
int[] somma = new int[lughezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];

//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = C.generaStringa(vett1);
String strVett2 = C.generaStringa(vett2);
String strSomma = C.generaStringa(somma);
```


SommaVettori (5): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = C.leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = C.leggiVettore(in, lunghezza);

//calcola il vettore somma
int[] somma = new int[lughezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];

//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = C.generaStringa(vett1);
String strVett2 = C.generaStringa(vett2);
String strSomma = C.generaStringa(somma);

//scrivi le stringhe
out.println("Vettore 1: [" + strVett1 + "]");
out.println("Vettore 2: [" + strVett2 + "]");
out.println("Vettore 3: [" + strSomma + "]");
```

Definizione dei sottoprogrammi

- Dato che non disponiamo della classe `C` con i due metodi `leggiVettore` e `generaStringa` e dobbiamo scrivere noi il codice di tali metodi.

Definizione dei sottoprogrammi

- ▶ Dato che non disponiamo della classe `C` con i due metodi `leggiVettore` e `generaStringa` e dobbiamo scrivere noi il codice di tali metodi.
- ▶ Poiché i due metodi servono essenzialmente per semplificare il metodo `main` di `SommaVettori` li definiamo come metodi `statici` e `privati` classe `SommaVettori`.

Definizione dei sottoprogrammi

```
private static String generaStringa(int[] vettore) {  
    String risultato = "";  
    for (int i = 0; i < vettore.length; i++)  
        risultato += vettore[i] + (i < vettore.length - 1 ? " " : "");  
    return risultato;  
}
```

Definizione dei sottoprogrammi

```
private static String generaStringa(int[] vettore) {  
    String risultato = "";  
    for (int i = 0; i < vettore.length; i++)  
        risultato += vettore[i] + (i < vettore.length - 1 ? " " : "");  
    return risultato;  
}
```

- L'intestazione specifica che il metodo *statico* di nome *leggiVettore* riceve un argomento di tipo *int[]* e restituisce come valore un riferimento di tipo *String*.

Definizione dei sottoprogrammi

```
private static String generaStringa(int[] vettore) {  
    String risultato = "";  
    for (int i = 0; i < vettore.length; i++)  
        risultato += vettore[i] + (i < vettore.length - 1 ? " " : "");  
    return risultato;  
}
```

- ▶ L'intestazione specifica che il metodo *statico* di nome *leggiVettore* riceve un argomento di tipo *int[]* e restituisce come valore un riferimento di tipo *String*.
- ▶ Il parametro di nome *vettore*, è a tutti gli effetti una variabile utilizzabile all'interno del corpo del metodo.

Definizione dei sottoprogrammi

```
private static String generaStringa(int[] vettore) {  
    String risultato = "";  
    for (int i = 0; i < vettore.length; i++)  
        risultato += vettore[i] + (i < vettore.length - 1 ? " " : "");  
    return risultato;  
}
```

- ▶ L'intestazione specifica che il metodo *statico* di nome *leggiVettore* riceve un *argomento* di tipo *int[]* e *restituisce come valore* un riferimento di tipo *String*.
- ▶ Il parametro di nome *vettore*, è a tutti gli effetti una variabile utilizzabile all'interno del corpo del metodo.
- ▶ Quando il metodo viene invocato, il valore dell'espressione indicata al posto del parametro (detto anche *argomento*) viene *copiato nella variabile vettore* ed è quindi disponibile all'interno del corpo del metodo tramite tale variabile.

Definizione dei sottoprogrammi

```
private static String generaStringa(int[] vettore) {  
    String risultato = "";  
    for (int i = 0; i < vettore.length; i++)  
        risultato += vettore[i] + (i < vettore.length - 1 ? "  " : "");  
    return risultato;  
}
```

- ▶ L'intestazione specifica che il metodo *statico* di nome *leggiVettore* riceve un argomento di tipo *int[]* e restituisce come valore un riferimento di tipo *String*.
- ▶ Il parametro di nome *vettore*, è a tutti gli effetti una variabile utilizzabile all'interno del corpo del metodo.
- ▶ Quando il metodo viene invocato, il valore dell'espressione indicata al posto del parametro (detto anche *argomento*) viene copiato nella variabile *vettore* ed è quindi disponibile all'interno del corpo del metodo tramite tale variabile.
- ▶ Per restituire un risultato viene utilizzata l'istruzione *return* seguita dal risultato da restituire (nell'esempio un riferimento di tipo *String*).

Definizione dei sottoprogrammi

```
private static int[] leggiVettore(ConsoleInputManager input,
                                  int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

Definizione dei sottoprogrammi

```
private static int[] leggiVettore(ConsoleInputManager input,
                                  int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

- Il metodo ha due argomenti: il primo, di tipo `ConsoleInputManager`, è il canale di comunicazione per effettuare la lettura da tastiera, il secondo, `lung`, è la lunghezza del vettore da leggere.

Definizione dei sottoprogrammi

```
private static int[] leggiVettore(ConsoleInputManager input,
                                   int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

- ▶ Il metodo ha due argomenti: il primo, di tipo `ConsoleInputManager`, è il canale di comunicazione per effettuare la lettura da tastiera, il secondo, `lung`, è la lunghezza del vettore da leggere.
- ▶ Il metodo restituisce il riferimento all'array letto.

Definizione dei sottoprogrammi: variabili locali

```
private static int[] leggiVettore(ConsoleInputManager input,
                                int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

Definizione dei sottoprogrammi: variabili locali

```
private static int[] leggiVettore(ConsoleInputManager input,
                                  int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

- Le variabili dichiarate all'interno di un metodo (come `vettore` nell'esempio) prendono il nome di *variabili locali*.

Definizione dei sottoprogrammi: variabili locali

```
private static int[] leggiVettore(ConsoleInputManager input,
                                   int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

- ▶ Le variabili dichiarate all'interno di un metodo (come **vettore** nell'esempio) prendono il nome di *variabili locali*.
- ▶ Le variabili locali *esistono e sono utilizzabili* solo all'interno del metodo in cui sono dichiarate e forniscono un supporto all'attività del metodo.

Definizione dei sottoprogrammi: variabili locali

```
private static int[] leggiVettore(ConsoleInputManager input,
                                  int lung) {
    int[] vettore = new int[lung];
    for (int i = 0; i < vettore.length; i++)
        vettore[i] = input.readInt("Elemento " + i + "? ");
    return vettore;
}
```

- ▶ Le variabili dichiarate all'interno di un metodo (come **vettore** nell'esempio) prendono il nome di *variabili locali*.
- ▶ Le variabili locali *esistono e sono utilizzabili* solo all'interno del metodo in cui sono dichiarate e forniscono un supporto all'attività del metodo.
- ▶ Quando l'esecuzione del metodo termina, non hanno più alcuna ragione di esistere e, di conseguenza, vengono distrutte.

SommaVettori (6): nuova soluzione

```
//leggi la lunghezza dei vettori  
int lunghezza = in.readInt("Lunghezza dei vettori? ");
```


SommaVettori (6): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = leggiVettore(in, lunghezza);
```

SommaVettori (6): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = leggiVettore(in, lunghezza);
```

SommaVettori (6): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = leggiVettore(in, lunghezza);

//calcola il vettore somma
int[] somma = new int[lughezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];
```

SommaVettori (6): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = leggiVettore(in, lunghezza);

//calcola il vettore somma
int[] somma = new int[lughezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];

//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = generaStringa(vett1);
String strVett2 = generaStringa(vett2);
String strSomma = generaStringa(somma);
```

SommaVettori (6): nuova soluzione

```
//leggi la lunghezza dei vettori
int lunghezza = in.readInt("Lunghezza dei vettori? ");

//leggi il primo vettore
out.println("Lettura primo vettore");
int[] vett1 = leggiVettore(in, lunghezza);

//leggi il secondo vettore
out.println("Lettura secondo vettore");
int[] vett2 = leggiVettore(in, lunghezza);

//calcola il vettore somma
int[] somma = new int[lughezza];
for (int i = 0; i < somma.length; i++)
    somma[i] = vett1[i] + vett2[i];

//costruisci le stringhe che rappresentano i tre vettori
String strVett1 = generaStringa(vett1);
String strVett2 = generaStringa(vett2);
String strSomma = generaStringa(somma);

//scrivi le stringhe
out.println("Vettore 1: [" + strVett1 + "]");
out.println("Vettore 2: [" + strVett2 + "]");
out.println("Vettore 3: [" + strSomma + "]");
```

Si scriva un'applicazione che:

- ▶ letta una sequenza di stringhe terminata dall'inserimento della stringa vuota
- ▶ visualizzi le stringhe lette nell'ordine in cui sono state inserite

Gestione di una sequenza di oggetti

Si scriva un'applicazione che:

- ▶ letta una sequenza di stringhe terminata dall'inserimento della stringa vuota
- ▶ visualizzi le stringhe lette nell'ordine in cui sono state inserite

Schema della soluzione:

(1) Fase di lettura

l'applicazione acquisisce la **sequenza** di stringhe che l'utente inserisce

Gestione di una sequenza di oggetti

Si scriva un'applicazione che:

- ▶ letta una sequenza di stringhe terminata dall'inserimento della stringa vuota
- ▶ visualizzi le stringhe lette nell'ordine in cui sono state inserite

Schema della soluzione:

(1) Fase di lettura

l'applicazione acquisisce la **sequenza** di stringhe che l'utente inserisce

(2) Fase di visualizzazione

l'applicazione visualizza sul monitor tutte le stringhe della **sequenza**

Gestione di una sequenza di oggetti

Si scriva un'applicazione che:

- ▶ letta una sequenza di stringhe terminata dall'inserimento della stringa vuota
- ▶ visualizzi le stringhe lette nell'ordine in cui sono state inserite

Schema della soluzione:

(1) Fase di lettura

l'applicazione acquisisce la **sequenza** di stringhe che l'utente inserisce

(2) Fase di visualizzazione

l'applicazione visualizza sul monitor tutte le stringhe della **sequenza**

Osservazione

Gli array non sono la struttura più adeguata per memorizzare la sequenza di stringhe in quanto non siamo in grado di fare previsioni sul numero di stringhe che verranno inserite.

Contratto

Le sue istanze rappresentano sequenze di oggetti di un tipo **E**, cioè collezioni di oggetti che possono contenere duplicazioni.

Nella sequenza gli oggetti compaiono nell'ordine in cui sono stati inseriti.

Contratto

Le sue istanze rappresentano sequenze di oggetti di un tipo **E**, cioè collezioni di oggetti che possono contenere duplicazioni.

Nella sequenza gli oggetti compaiono nell'ordine in cui sono stati inseriti.

Le classi e i tipi generici:

- ▶ Indicati nella documentazione con una notazione del tipo **Sequenza<E>**

Contratto

Le sue istanze rappresentano sequenze di oggetti di un tipo **E**, cioè collezioni di oggetti che possono contenere duplicazioni.

Nella sequenza gli oggetti compaiono nell'ordine in cui sono stati inseriti.

Le classi e i tipi generici:

- ▶ Indicati nella documentazione con una notazione del tipo Sequenza<E>
- ▶ **E** viene detto **tipo parametro**

Contratto

Le sue istanze rappresentano sequenze di oggetti di un tipo **E**, cioè collezioni di oggetti che possono contenere duplicazioni.

Nella sequenza gli oggetti compaiono nell'ordine in cui sono stati inseriti.

Le classi e i tipi generici:

- ▶ Indicati nella documentazione con una notazione del tipo **Sequenza<E>**
- ▶ **E** viene detto **tipo parametro**
- ▶ Possiamo dire che la classe o il tipo è “**Sequenza di E**”

Creazione di un oggetto di una classe generica

```
new Sequenza<String>()
```

Creazione di un oggetto di una classe generica

```
new Sequenza<String>()
```



È un oggetto in grado di memorizzare una sequenza di stringhe

Creazione di un oggetto di una classe generica

```
new Sequenza<String>()
```



È un oggetto in grado di memorizzare una **sequenza di stringhe**

```
new Sequenza<Frazione>()
```


Creazione di un oggetto di una classe generica

```
new Sequenza<String>()
```



È un oggetto in grado di memorizzare una **sequenza di stringhe**

```
new Sequenza<Frazione>()
```



È un oggetto in grado di memorizzare una **sequenza di frazioni**

Memorizzazione di un oggetto di un tipo generico

```
Sequenza<String> seq = new Sequenza<String>()
```

Memorizzazione di un oggetto di un tipo generico

```
Sequenza<String> seq = new Sequenza<String>()
```

```
Sequenza<Frazione> seq = new Sequenza<Frazione>()
```

Memorizzazione di un oggetto di un tipo generico

```
Sequenza<String> seq = new Sequenza<String>()
```

```
Sequenza<Frazione> seq = new Sequenza<Frazione>()
```

I tipi `Sequenza<String>` e `Sequenza<Frazione>` vengono detti **tipi parametrizzati**

Costruttori

- ▶ `public Sequenza()`
Costruisce una sequenza vuota.

Metodi

- ▶ `public boolean add(E o)`

Aggiunge alla fine della sequenza l'oggetto fornito tramite l'argomento e restituisce `true`.

Nel caso come argomento venga fornito `null`, non modifica la sequenza e restituisce `false`.

Metodi

- ▶ `public boolean add(E o)`

Aggiunge alla fine della sequenza l'oggetto fornito tramite l'argomento e restituisce `true`.

Nel caso come argomento venga fornito `null`, non modifica la sequenza e restituisce `false`.

- ▶ `public int size()`

Restituisce il numero di elementi presenti nella sequenza.

Metodi

► `public boolean add(E o)`

Aggiunge alla fine della sequenza l'oggetto fornito tramite l'argomento e restituisce `true`.

Nel caso come argomento venga fornito `null`, non modifica la sequenza e restituisce `false`.

► `public int size()`

Restituisce il numero di elementi presenti nella sequenza.

► `public boolean isEmpty()`

Restituisce `true` se e solo se la sequenza è vuota.

Metodi

- ▶ `public boolean contains(E o)`

Restituisce true se e solo se la sequenza contiene un oggetto uguale (sulla base del criterio di uguaglianza fornito dal metodo `equals`) a quello specificato tramite l'argomento.

Metodi

- ▶ `public boolean contains(E o)`

Restituisce true se e solo se la sequenza contiene un oggetto uguale (sulla base del criterio di uguaglianza fornito dal metodo `equals`) a quello specificato tramite l'argomento.

- ▶ `public E find(E o)`

Restituisce il riferimento al primo oggetto nella sequenza uguale a quello specificato tramite l'argomento, o `null` se tale oggetto non è presente.

Metodi

▶ `public boolean contains(E o)`

Restituisce `true` se e solo se la sequenza contiene un oggetto uguale (sulla base del criterio di uguaglianza fornito dal metodo `equals`) a quello specificato tramite l'argomento.

▶ `public E find(E o)`

Restituisce il riferimento al primo oggetto nella sequenza uguale a quello specificato tramite l'argomento, o `null` se tale oggetto non è presente.

▶ `public boolean remove(E o)`

Elimina dalla sequenza il primo oggetto uguale a quello specificato tramite l'argomento e restituisce `true`. Nel caso tale oggetto non vi sia, lascia la sequenza immutata e restituisce `false`.

```
...  
//predisposizione della "memoria"  
Sequenza<String> memo = new Sequenza<String>();
```

```
...  
//predisposizione della "memoria"  
Sequenza<String> memo = new Sequenza<String>();  
  
//fase di lettura  
String s = in.readLine();  
while (!s.equals("")) {  
    memo.add(s);  
    s = in.readLine();  
}
```

```
...
//predisposizione della "memoria"
Sequenza<String> memo = new Sequenza<String>();

//fase di lettura
String s = in.readLine();
while (!s.equals("")) {
    memo.add(s);
    s = in.readLine();
}

//visualizzazione della sequenza

...per ogni elemento della sequenza
    visualizzalo
...
```

È possibile scorrere gli elementi contenuti in un oggetto di tipo `Sequenza<E>`, dal primo all'ultimo, utilizzando un ciclo `for-each`.

Sequenza e ciclo *for-each*

È possibile scorrere gli elementi contenuti in un oggetto di tipo `Sequenza<E>`, dal primo all'ultimo, utilizzando un ciclo *for-each*.

```
Sequenza<E> sequenza;  
...  
for (E elemento: sequenza)  
    ...usa elemento...
```


Sequenza e ciclo *for-each*

È possibile scorrere gli elementi contenuti in un oggetto di tipo `Sequenza<E>`, dal primo all'ultimo, utilizzando un ciclo *for-each*.

```
Sequenza<E> sequenza;  
...  
for (E elemento: sequenza)  
    ...usa elemento...
```

Esempio

```
Sequenza<String> memo = new Sequenza<String>();  
...  
//fase di scrittura  
for (String x : memo)  
    out.println(x);
```

```
...
//predisposizione della "memoria"
Sequenza<String> memo = new Sequenza<String>();

//fase di lettura
String s = in.readLine();
while (!s.equals("")) {
    memo.add(s);
    s = in.readLine();
}

//fase di scrittura
for (String x : memo)
    out.println(x);
...
```

La classe generica SequenzaOrdinata<E>

Contratto

Le sue istanze rappresentano sequenze ordinate di oggetti di tipo E.

La classe generica SequenzaOrdinata<E>

Contratto

Le sue istanze rappresentano sequenze ordinate di oggetti di tipo E.

Ad esempio:

- ▶ Se E è il tipo `String`
la sequenza è ordinata secondo l'ordine alfabetico

La classe generica SequenzaOrdinata<E>

Contratto

Le sue istanze rappresentano sequenze ordinate di oggetti di tipo E.

Ad esempio:

- ▶ Se E è il tipo `String`
la sequenza è ordinata secondo l'ordine alfabetico
- ▶ Se E è il tipo `Frazione` o il tipo `Integer`
la sequenze è ordinata in maniera crescente

La classe generica SequenzaOrdinata<E>

Contratto

Le sue istanze rappresentano sequenze ordinate di oggetti di tipo E.

Ad esempio:

- ▶ Se E è il tipo `String`
la sequenza è ordinata secondo l'ordine alfabetico
- ▶ Se E è il tipo `Frazione` o il tipo `Integer`
la sequenza è ordinata in maniera crescente
- ▶ Se E è il tipo `Data`
la sequenza è ordinata cronologicamente

```
...  
//predisposizione della "memoria"  
SequenzaOrdinata<String> memo = new SequenzaOrdinata<String>();
```

```
...  
//predisposizione della "memoria"  
SequenzaOrdinata<String> memo = new SequenzaOrdinata<String>();  
  
//fase di lettura  
String s = in.readLine();  
while (!s.equals("")) {  
    memo.add(s);  
    s = in.readLine();  
}
```



```
...
//predisposizione della "memoria"
SequenzaOrdinata<String> memo = new SequenzaOrdinata<String>();

//fase di lettura
String s = in.readLine();
while (!s.equals("")) {
    memo.add(s);
    s = in.readLine();
}

//fase di scrittura
for (String x : memo)
    out.println(x);
...
```

- È possibile costruire sequenze di oggetti di un qualunque tipo

Possiamo usare qualunque tipo riferimento per istanziare il tipo parametro **E** di **Sequenza<E>**

- ▶ È possibile costruire sequenze di oggetti di un qualunque tipo

Possiamo usare qualunque tipo riferimento per istanziare il **tipo parametro E** di **Sequenza<E>**

- ▶ Per costruire sequenze ordinate è **necessario** che il tipo degli elementi sia **“ordinabile”**, cioè che sia definita una relazione di ordine totale tra i suoi elementi.

Possiamo usare solo tipi riferimento su cui sia definita una relazione di **ordine totale** per istanziare il **tipo parametro E** di **SequenzaOrdinata<E>**

- ▶ È possibile costruire sequenze di oggetti di un qualunque tipo

Possiamo usare qualunque tipo riferimento per istanziare il **tipo parametro E** di **Sequenza<E>**

- ▶ Per costruire sequenze ordinate è **necessario** che il tipo degli elementi sia **“ordinabile”**, cioè che sia definita una relazione di ordine totale tra i suoi elementi.

Possiamo usare solo tipi riferimento su cui sia definita una relazione di **ordine totale** per istanziare il **tipo parametro E** di **SequenzaOrdinata<E>**

- ▶ Esistono quindi delle **limitazioni** sulla genericità della classe **SequenzaOrdinata** che studieremo nel seguito