Le collezioni di Valori - Insiemi e Strutture Dati per gestirle

Strutture Dati in Java

Vettori, Liste e operazioni Map / Filter / Reduce

Fino a questo momento hai lavorato soprattutto con **singoli oggetti**: un numero, una stringa, una persona. È un passaggio fondamentale per imparare le basi, ma nei programmi reali succede raramente di gestire una sola cosa alla volta.

Nella pratica lavoriamo quasi sempre con **insiemi di dati**: più numeri, più nomi, più utenti, più record.

Per gestire questi insiemi in modo ordinato ed efficiente, Java mette a disposizione le **strutture dati**.

Cos’è una struttura dati

Una **struttura dati** è un insieme di variabili collegate tra loro, pensato per essere trattato come un’unica entità.

Non ragioniamo più sul singolo valore, ma sull’insieme nel suo complesso, seguendo regole precise su:

- come si memorizzano i dati
- come si accede agli elementi
- come si scorre l’insieme
- quali operazioni sono permesse

La prima struttura dati che si incontra in Java è il **vettore (array)**.

Vettori (Array)

Cos'è un vettore

Un **vettore** è:

- una **lista ordinata**
- di elementi **tutti dello stesso tipo**
- con **dimensione fissa**

Ordinata significa che ogni elemento ha una **posizione precisa**, chiamata **indice**.

Gli indici partono sempre da **0**.

I tre momenti fondamentali del vettore

Pensiamo al vettore come a un **palazzo con appartamenti numerati**.

Dichiarazione – "Annuncio il palazzo"

Dici a Java che esisterà un vettore, ma **il palazzo non è ancora costruito**.

```
int[] numeri;
```

- `int[]` → tipo degli elementi
- `numeri` → nome della variabile
- **Il vettore NON esiste ancora**: non puoi usarlo

Creazione – "Costruisco il palazzo"

Ora decidi **quanti appartamenti** avrà il palazzo.

```
numeri = new int[5];
```

- viene creato un vettore di 5 elementi

- gli indici vanno da `0` a `4`
- ogni elemento ha un **valore di default** (`0` per gli int)

Inizializzazione – “Arredo gli appartamenti”

Inserisci i valori nelle singole posizioni.

```
numeri[0] = 10;   // assegna 10 alla posizione 0
numeri[1] = 20;   // assegna 20 alla posizione 1
```

Se accedi a una posizione che non esiste, il programma va in errore (`ArrayIndexOutOfBoundsException`).

Scorrere un vettore

Cos’è lo scorrimento

Scorrere un vettore significa visitare tutti gli elementi, uno alla volta, dalla prima all’ultima posizione.

È una delle operazioni più importanti, perché consente di:

- stampare tutti i valori
- sommare i numeri
- contare elementi che rispettano una condizione
- trovare massimo o minimo

Esempio: somma degli elementi

```
int somma = 0;

for (int i = 0; i < numeri.length; i++) {
    somma += numeri[i];   // aggiunge ogni elemento alla somma
}
```

- `numeri.length` → dimensione del vettore
- il ciclo parte da `0` e arriva all'ultima posizione valida

Cambiare mentalità

Quando lavori con un insieme di dati, nascono tre idee chiave:

- **trasformare** un insieme in un altro
- **selezionare** solo alcuni elementi
- **ridurre** l'insieme a un solo valore

Queste idee si chiamano: **Map, Filter e Reduce**.

Map, Filter e Reduce

Map (Mappatura)

A cosa serve

Trasforma **ogni elemento** di un insieme nello stesso modo.

- **Input**: un insieme
- **Output**: un nuovo insieme **della stessa dimensione**

Esempio

```
Teacher[] insegnanti = ...;              // insieme di partenza
String[] nomi = new String[insegnanti.length]; // stesso numero di elementi

for (int i = 0; i < insegnanti.length; i++) {
   nomi[i] = insegnanti[i].getName()
         + " "
         + insegnanti[i].getSurname(); // trasformazione
}
```

Filter (Filtro)

A cosa serve

Seleziona **solo alcuni elementi** che rispettano una condizione.

- **Input**: un insieme
- **Output**: un insieme più piccolo (o vuoto)

Esempio

```
Person[] persone = ...;                 // insieme di partenza
List<Person> donne = new ArrayList<>();   // lista inizialmente vuota

for (Person p : persone) {
    if (p.getGender().equals("F")) {
        donne.add(p);                   // aggiunge solo chi soddisfa la condizione
    }
}
```

Reduce (Riduzione)

A cosa serve

“Comprime” un insieme in **un solo valore**.

- **Input**: molti elementi
- **Output**: un valore singolo

Esempio

```
int costoTotale = 0;

for (Person p : persone) {
```

```
    costoTotale += p.getCost();  // accumula i valori
}
```

Le Liste (ArrayList)

Perché servono

Il limite principale dei vettori è che **la dimensione è fissa**.

Le **liste** risolvono il problema perché:

- nascono vuote
- crescono e si restringono automaticamente

Creazione di una lista

```
List<String> nomi = new ArrayList<>();
```

- `List<String>` → interfaccia
- `ArrayList` → implementazione concreta
- la lista parte con **0 elementi**

Operazioni principali

Aggiungere elementi

```
nomi.add("Giorgio");
nomi.add("Chiara");
```

non devi indicare la posizione

Leggere un elemento

```
String primo = nomi.get(0);
```

Numero di elementi

```
int quanti = nomi.size();
```

Confronto: Vettori vs Liste

Caratteristica	Vettori (Array)	Liste (ArrayList)
Dimensione	Fissa	Dinamica
Tipo	Primitivi e oggetti	Solo oggetti
Lunghezza	`length`	`size()`
Aggiunta elementi	Manuale	Automatica
Flessibilità	Bassa	Alta

Tipi Boxati (Wrapper)

Perché servono

Le liste possono contenere **solo oggetti**, non tipi primitivi.

`List<int>`

`List<Integer>`

Tipi principali

Primitivo	Boxato
int	Integer
double	Double
boolean	Boolean
char	Character

Autoboxing e Unboxing

Java gestisce automaticamente la conversione:

- **Autoboxing**: `int → Integer`
- **Unboxing**: `Integer → int`

Esempio

```
List<Integer> numeri = new ArrayList<>();

numeri.add(5);       // autoboxing: int → Integer
numeri.add(10);

int x = numeri.get(0); // unboxing: Integer → int
```

Quick Reference – Map, Filter, Reduce

Operazione	Input	Output	Scopo
Map	Insieme	Insieme (stessa dimensione)	Trasformare
Filter	Insieme	Insieme (più piccolo)	Selezionare
Reduce	Insieme	Valore singolo	Riassumere