Gli ArrayList

Docente
Mario Perna
prof.perna.mario@darzo.net

A.S.
2025/2026
Materia
Informatica

Introduzione

Gli array tradizionali in Java hanno una limitazione: **hanno dimensione fissa**.

Una volta creati, non possono crescere o ridursi.

Gli **ArrayList** risolvono proprio questo problema: sono **collezioni dinamiche** che possono espandersi o contrarsi durante l'esecuzione del programma, mantenendo al tempo stesso tutta la semplicità dell'accesso tramite indice.

Cos’è un ArrayList

Un ArrayList è una **classe** del pacchetto **java.util** che **implementa l’interfaccia List**.

Si comporta come un array, ma con capacità di **ridimensionarsi automaticamente**.

È ideale quando non conosciamo a priori la quantità di dati che dovremo gestire.

In altre parole, è un **contenitore dinamico** in cui possiamo aggiungere, modificare o rimuovere elementi in modo semplice.

Come si crea un ArrayList

Per poterlo usare, bisogna importare la classe:

```
import java.util.ArrayList;
```

Poi possiamo creare un ArrayList specificando il tipo di dati che conterrà:

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

In questo esempio, l'ArrayList conterrà solo stringhe.

L'uso dei **generics (<String>)** permette di garantire che tutti gli elementi siano dello stesso tipo, evitando errori di tipo a tempo di compilazione.

Aggiungere, leggere e modificare

L’ArrayList mette a disposizione metodi molto intuitivi:

```
nomi.add("Alice");   // Aggiunge un elemento
String primo = nomi.get(0);   // Legge il primo elemento
nomi.set(0, "Bob");   // Modifica il primo elemento
```

Gli indici partono da 0, come negli array.

Queste operazioni sono rapide e dirette, perfette per gestire liste di dati in modo dinamico.

Rimuovere elementi

Possiamo rimuovere un elemento in due modi:

```
nomi.remove(0);       // Rimuove l'elemento in posizione 0
nomi.remove("Alice"); // Rimuove l'oggetto "Alice"
```

Se ci sono duplicati, il metodo **remove()** elimina solo la **prima occorrenza** trovata.

Se l'elemento non è presente, non accade nulla di grave: l'ArrayList rimane invariato.

Scorrere un ArrayList

Per leggere tutti gli elementi di un ArrayList possiamo usare un ciclo for o, più comunemente, un **foreach**:

```
for (String nome : nomi) {
        System.out.println(nome);
}
```

Questo approccio è elegante e leggibile, e permette di concentrarsi sul contenuto della lista senza preoccuparsi degli indici.

Metodi utili

Gli ArrayList offrono tanti metodi pronti all'uso:

- **size()** → restituisce la dimensione della lista
- **contains("Alice")** → verifica se un elemento è presente
- **isEmpty()** → controlla se la lista è vuota
- **clear()** → cancella tutti gli elementi

Questi metodi semplificano molto la gestione dei dati, rendendo l'ArrayList uno strumento versatile e potente.

Vantaggi

Gli ArrayList sono estremamente **flessibili**:

- non serve conoscere la dimensione in anticipo;
- la gestione degli elementi è semplice e intuitiva;
- fanno parte del **framework delle collezioni di Java**, quindi integrano bene con altre strutture dati come Set e Map.

Inoltre, essendo molto usati, hanno un comportamento prevedibile e una documentazione ampia.

Limiti e attenzioni

Tuttavia, non sono sempre la scelta migliore.

- Le **inserzioni o rimozioni nel mezzo** della lista possono essere costose, perché tutti gli elementi successivi devono essere spostati.
- Non sono **thread-safe**, quindi in ambienti multithread bisogna usare alternative come CopyOnWriteArrayList o Collections.synchronizedList().

Per liste con modifiche frequenti, una LinkedList può essere più efficiente.

Quando usarli

Usa un **ArrayList** quando:

- ti serve una **lista dinamica** di elementi dello stesso tipo;
- devi **accedere frequentemente** agli elementi per indice;
- vuoi sfruttare i metodi già pronti del framework delle collezioni.

Evita invece di usarlo quando le operazioni di inserimento o rimozione in posizioni intermedie sono molto frequenti.