Set e Map

Perché servono Set e Map?

Quando si lavora con la programmazione, raramente ci si trova a gestire un singolo dato isolato. Nella maggior parte dei casi è necessario organizzare e manipolare insiemi di informazioni, che possono rappresentare nomi, ruoli, interessi o qualsiasi altro tipo di dato. Java offre diversi tipi di contenitori per organizzare questi dati, ciascuno progettato per rispondere a esigenze specifiche. Tra questi, Set e Map rappresentano due strutture fondamentali che risolvono problemi di natura differente

Set: quando NON vuoi duplicati

Il Set è una struttura dati pensata per gestire collezioni di elementi in cui non sono ammesse ripetizioni. La caratteristica distintiva di un Set è proprio questa: ogni elemento può comparire una sola volta all'interno della collezione. Questo lo rende particolarmente utile quando si desidera rappresentare insiemi nel senso matematico del termine, dove la duplicazione non ha senso logico. Se si tenta di aggiungere un elemento già presente nel Set, l'operazione non produce alcun effetto e la dimensione della collezione rimane invariata.

È importante comprendere che un Set non garantisce alcun ordinamento degli elementi. Non esiste un concetto di "primo elemento" o "ultimo elemento" in senso posizionale. Gli elementi sono semplicemente contenuti nella struttura senza una sequenza predefinita. Di conseguenza, non è possibile accedere agli elementi tramite un indice numerico come avviene con le liste. Per attraversare un Set è necessario utilizzare meccanismi di iterazione che non fanno affidamento sulla posizione, ma che semplicemente visitano tutti gli elementi presenti.

A cosa serve un Set?

Un **Set** serve quando vuoi **tenere un insieme di elementi senza ripetizioni**.

Esempio reale:

- interessi di una persona

- numeri vincenti del lotto
- ruoli unici in una squadra

Se un elemento è già presente, **non viene aggiunto di nuovo**

- Un Set **non accetta duplicati**
- **Non garantisce l'ordine** degli elementi
- Puoi aggiungere e togliere elementi
- Non puoi dire "dammi il primo", perché **non esiste il concetto di primo**

```
Set<String> interests = new HashSet<>();

interests.add("G1");
interests.add("G1"); // duplicato
interests.add("A");


Console.print(interests.size()); // stampa 2
```

Scorrere un Set

Dato che **non c'è ordine**, lo si scorre così:

```
for (String i : interests) {
    Console.print(i);
}
```

Metodo molto utile: `contains`

Serve a chiedere:

> "Questo elemento c'è oppure no?"

```
interests.contains("A"); // true
interests.contains("B"); // false
```

Una delle operazioni più significative che un Set permette di eseguire in modo efficiente è la verifica di appartenenza. Attraverso il metodo contains è possibile determinare rapidamente se un determinato elemento è presente o meno nella collezione. Questa funzionalità rappresenta uno dei principali vantaggi nell'utilizzo di un Set rispetto ad altre strutture dati.

Map: quando vuoi collegare una cosa a un’altra

La Map, invece, risponde a un'esigenza completamente diversa. Mentre il Set gestisce semplici collezioni di elementi, la Map permette di stabilire associazioni tra coppie di dati. Ogni elemento di una Map è costituito da una chiave e da un valore corrispondente. La chiave funge da identificatore univoco attraverso cui è possibile recuperare il valore associato. Il concetto è analogo a quello di una rubrica telefonica, dove il nome della persona rappresenta la chiave e il numero di telefono costituisce il valore.
La caratteristica fondamentale delle chiavi in una Map è la loro unicità. Non possono esistere due chiavi identiche all'interno della stessa Map. Se si tenta di inserire una coppia con una chiave già presente, il valore precedentemente associato a quella chiave viene sostituito dal nuovo valore. I valori, al contrario, possono ripetersi liberamente. Più chiavi diverse possono essere associate allo stesso valore senza alcun vincolo.

A cosa serve una Map?

Una **Map** serve quando vuoi **associare una chiave a un valore**.

Esempi reali:

- `"nome" → "George"`
- `"età" → 40`
- `"ruolo" → Person"`

Pensa a una **rubrica**:

- il nome è la **chiave**
- il numero di telefono è il **valore**

Le chiavi NON si possono ripetere

I valori SÌ

Come funziona

- Ogni elemento è una **coppia**: chiave + valore
- Per inserire usi `put`
- Per leggere usi `get(chiave)`

```
Map<String, String> p = new HashMap<>();

p.put("name", "George");
p.put("surname", "Romano");
p.put("age", "40");
Console.print(p.get("age")); // stampa 40
```

Nota importante:

- Con le **liste** usi un numero (`get(0)`)
- Con le **Map** usi la **chiave**

L'accesso ai dati in una Map avviene specificando la chiave. Attraverso il metodo get, fornendo una chiave come parametro, si ottiene il valore ad essa associato. Questo meccanismo di accesso differisce profondamente da quello delle liste, dove si utilizza un indice numerico per identificare la posizione dell'elemento. Nelle Map non esiste il concetto di posizione sequenziale: l'accesso è basato esclusivamente sulla corrispondenza tra chiavi e valori.
L'inserimento di nuove coppie chiave-valore avviene tramite il metodo put, che accetta come parametri prima la chiave e poi il valore da associare. Questa operazione permette di costruire gradualmente la struttura dati, aggiungendo le associazioni necessarie in base alle esigenze dell'applicazione. La Map rappresenta quindi uno strumento potente per modellare relazioni tra dati, permettendo di organizzare informazioni in modo strutturato e facilmente accessibile attraverso identificatori significativi.