Iteratori / 11-03

Si vuole rispondere in modo efficiente e generale (indipendente dalla collezione, sia essa di alberi, liste etc...) a problemi come
→ Ricerca di primi elementi in una lista
→ somma gli elementi di un set / lista
→ trova il max di un set / lista

Si introduce il DESIGN PATTERN: uno schema risolutivo x risolvere problemi in modo efficiente

Gli iteratori possono essere INTERNI (verso la programmazione funzionale) ed ESTERNI (verso programmazione ad oggetti) con i loro vantaggi e svantaggi.

- Oggetto Iterabile: object container dei dati da iterare
- Iteratori su ↑ : oggetto che itera sugli elementi di ↑

Posso avere più iteratori per oggetto iterabile.

Esempio

Implementation of `search` with the iterator pattern

```
public static int search(int e, List<Integer> ls) {
    int res = 0;
    Iterator<Integer> it = ls.iterator(); // returns a new iterator on ls
    while (it.hasNext()) {
        if (it.next() == e) return res;
        res++;
    }
    return -1;
}

// more concise version with the enhanced for (for-each)
public static int search(int e, List<Integer> ls) {
    int res = 0;
    for (int el : ls) { // for-each with iterable objects or arrays
        if (el == e) return res;
        res++;
    }
    return -1;
}
```

→ iteratore implicito

Var

Parola chiave che permette di specificare variabili locali il cui tipo è dedotto dal compilatore automaticamente (PERMESSO solo x var locali)

Se uso var devo inizializzare la variabile.

Examples of `var` declarations

```
var r = new Range(2); // inferred type: Range
var it = r.iterator(); // inferred type: RangeIterator
var el = it.next(); // inferred type: Integer
// inferred type for s: HashSet<Integer>
var s = new HashSet<>(Arrays.asList(new Integer[] { 1, 2, 3, 4 }));
```

Metodi 'default'

Possono essere definiti all'interno delle interfacce.

- Sono metodi non astratti
- Solitamente usati x lanciare eccezioni
- Spesso opzionali

Example of use: definition of *optional* methods

```
public interface Iterator<E> {
    boolean hasNext();
    E next();
    // optional method, by default it throws UnsupportedOperationException
    default void remove() {
        throw new UnsupportedOperationException();
    }
    ...
    var r = new Range(2);
    var it = r.iterator();
    it.next();
    it.remove(); // throws UnsupportedOperationException
}
```