Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

Università degli Studi dell'Aquila

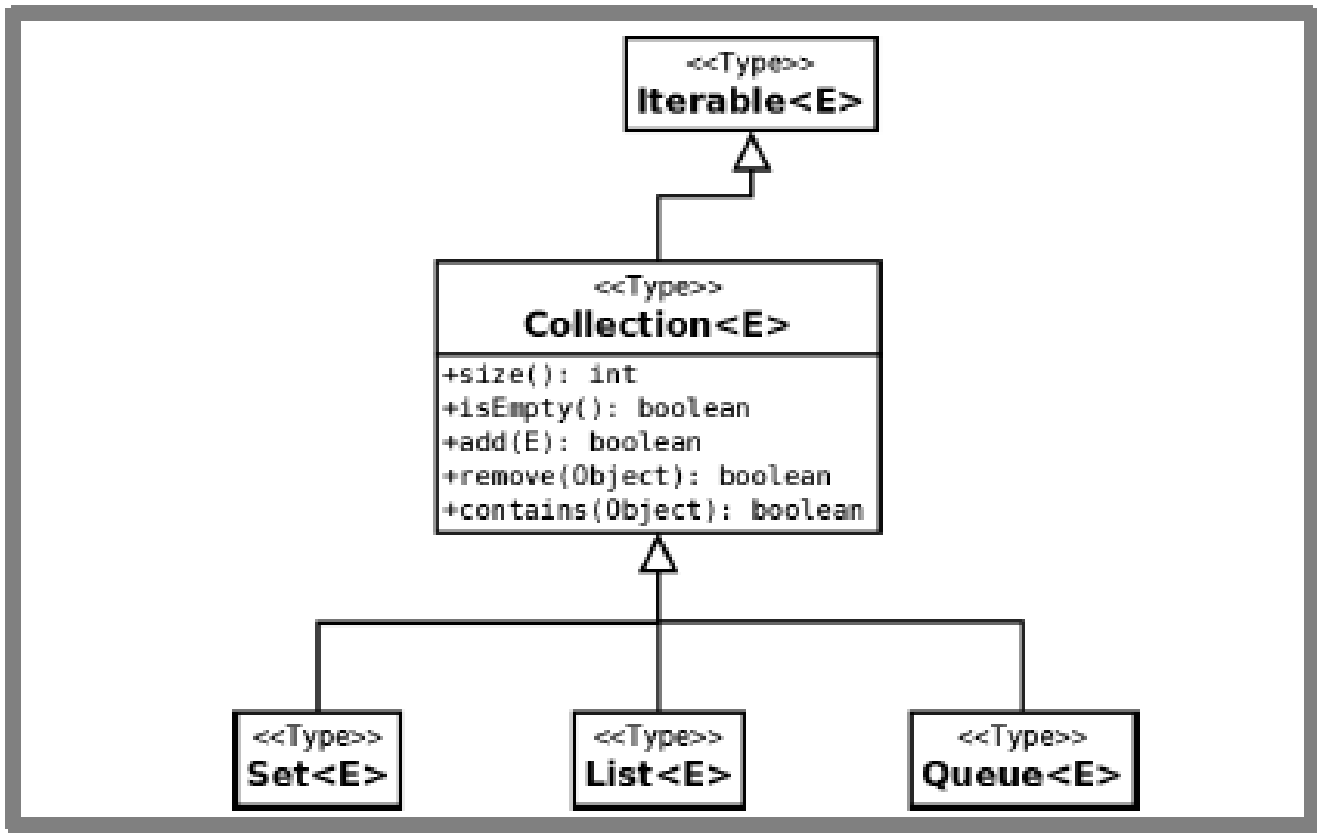
Corso di Algoritmi e Strutture Dati con Laboratorio

La libreria standard: JCF (continua)

L'interfaccia Iterable

`public interface Collection<E> extends Iterable<E>`

- Interfacce collegate con `Collection`:



Iteratori

- Consideriamo i seguenti esempi:
  - Dato un oggetto di una classe che implementa l'interfaccia `Collection` ("oggetto Collection" in breve) di studenti, visualizzare gli studenti migliori
  - Dato un oggetto `Collection` di membri di un club, aggiornare le quote dovute da ciascuno di essi
  - Dato un oggetto `Collection` di dipendenti a tempo pieno, calcolare il loro salario medio
- Notiamo che in ciascun esempio il compito da svolgere richiede l'accesso a tutti gli elementi di un oggetto `Collection`, uno dopo l'altro.

Iteratori

- Come si può fare in modo che qualsiasi implementazione dell'interfaccia `Collection` consenta ai suoi utilizzatori di eseguire un'iterazione che coinvolga, uno dopo l'altro, tutti gli elementi presenti in un suo esemplare, senza violare il principio di astrazione per i dati?
- La soluzione risiede nell'uso degli **iteratori,** oggetti che consentono di accedere agli elementi di oggetti `Collection`.

Iteratori: interfacce `Iterable<E>` e `Iterator<E>`

```
public interface Iterable<E> {
  public Iterator<E> iterator();
}

public interface Iterator<E> {
  public E next();
  public boolean hasNext();
  public void remove();  //optional
}
```

- Ogni classe che implementa `Iterable<E>` deve avere un metodo `iterator()` che restituisce un iteratore sugli elementi interni alla classe stessa

Iteratori: l'interfaccia `Iterator<E>`

- L'interfaccia **`Iterator<E>`** astrae il processo di scandire gli elementi di una collezione uno alla volta
- Permette di scandire gli elementi della struttura dati a prescindere dall'implementazione della struttura dati
- In Java un iteratore ha due primitive fondamentali specificate dall'interfaccia **`java.util.Iterator`**:

1. **`hasNext()`**: verifica se c'è ancora un elemento nella collezione
2. **`next()`**: restituisce il prossimo elemento della collezione

Iteratori: un esempio

- Sia `myColl` un riferimento ad un esemplare di una classe `Collection<String>`. Si vogliono visualizzare tutti i suoi elementi che iniziano con la lettera 'a'.
- Creiamo un oggetto iteratore. Un iteratore si ottiene invocando il metodo `iterator()` sull'oggetto che rappresenta la collezione stessa:

```
Collection<String> myColl = new …;
…
Iterator<String> itr = myColl.iterator();
```

Iteratori: un esempio

```
Iterator<String> itr = myColl.iterator();
```

- Eseguiamo la scansione:

```
String word;
while (itr.hasNext()){
  word=itr.next();
  if (word.charAt(0)==‘a’)
     System.out.println(word); }
```

Iteratori: schema tipico

```
Iterator<T> it = ottieni un iteratore
                 per la collezione
while (it.hasNext()){
  T elem = it.next();
  elabora l'elemento
}
```

Iteratori: il problema dei duplicati

```
public static boolean verificaDupOrdIterator(List<String> S){
  Collections.sort(S);  //ordina la lista di stringhe
  Iterator<String> it = S.iterator();
  if (!it.hasNext()) return false;
  String pred = it.next();
  while (it.hasNext()){
     String succ=it.next();
     if (pred.equals(succ)) return true;
     pred=succ;
  }
  return false;
}
```

Il ciclo for-each

- Se un oggetto `myColl` appartiene ad una collection class che implementa `Iterable<A>`, per una data classe `A`, è possibile scrivere il seguente ciclo for-each:

```
for (A a: myColl) {
// corpo del ciclo
...
}
```

Il ciclo for-each

- Il ciclo precedente è equivalente al blocco seguente:

```
Iterator<A> it = myColl.iterator();
while (it.hasNext()) {
A a = it.next();
// corpo del ciclo
...
}
```

- Come si vede, il ciclo for-each è più sintetico e riduce drasticamente il rischio di scrivere codice errato

Il ciclo `for-each`

- Il ciclo for-each:

```
for (A a: <exp>) {
    // corpo del ciclo
    ...
}
```

- è corretto a queste condizioni:

1. <exp> è una espressione di tipo “array di T” oppure di un sottotipo di “Iterable<T>”
2. T è assegnabile ad A

Il ciclo for-each

- Esempio: sia `myColl` un riferimento ad un esemplare di una classe `Collection<String>`. Si vogliono visualizzare tutti i suoi elementi che iniziano con la lettera 'a'.

- "*for each word in myColl…*"

```
for (String word: myColl)
    if (word.charAt(0)=='a')
      System.out.println(word);
```

Iteratori

- **`void remove()`** removes from the underlying collection the last element returned by this iterator (optional operation).
- This method can be called only once per call to **`next()`**.
- The behavior of an iterator is unspecified if the underlying collection is modified while the iteration is in progress in any way other than by calling this method.

Esempio: la classe **`RandomList2`**

- Avremmo potuto usare un enunciato for-each avanzato per scandire `randList`? No, perché oltre a ispezionare la lista, eliminiamo alcuni elementi contenuti in essa.

Iteratori

Ricorda:

- Ogni classe per rappresentare collezioni di elementi dovrebbe implementare l'interfaccia `Iterable<E>`.
- L'interfaccia `List<E>` estende l'interfaccia `Iterable<E>`, pertanto ogni oggetto di tipo `List<E>` è "iterabile".