Collezioni

ALBERTO FERRARI

Le collezioni di oggetti

Una collezione può memorizzare un numero arbitrario di oggetti

Il numero di elementi di una collezione è variabile:

- È possibile inserire nuovi oggetti
- È possibile eliminare oggetti

Librerie di classi

Una delle catatteristiche dei linguaggi object oriented che li rende molto potenti è la presenza di librerie di classi

Le librerie tipicamente contengono decine o centinaia di classi utili per gli sviluppatori e utilizzabili in un ampio insieme di applicazioni

In Java le librerie vengono definite packages

Il package java.util contiene classi per la gestione di collezioni di oggetti

Java2 collection

Le collection di Java 2 consistono di:

- *interfacce*: tipi di dati astratti che rappresentano collezioni,
  - List, Queue, Set, Map
- parziali implementazioni di interfacce facilmente riadattabili
- implementazioni concrete: classi basilari che implementano le interfacce fondamentali
- algoritmi: implementazioni di funzioni basilari (ordinamento, ricerca) applicabili a tutte le collezioni

Caratteristiche comuni alle collezioni

Possibilità di aumentare la capacità (se necessario)

Mantenere un contatore privato del numero di oggetti presenti

- Il valore è accessibile mediante il metodo size()

Mantenere l'ordine di inserimento degli elementi

Ogni elemento ha un indice

- Il valore dell'indice può cambiare a causa di operazioni di inserimento o eliminazione

I dettagli implementativi sono «nascosti»

- È importante?
- Questo ci permette ugualmente di utilizzare le classi?

Collection

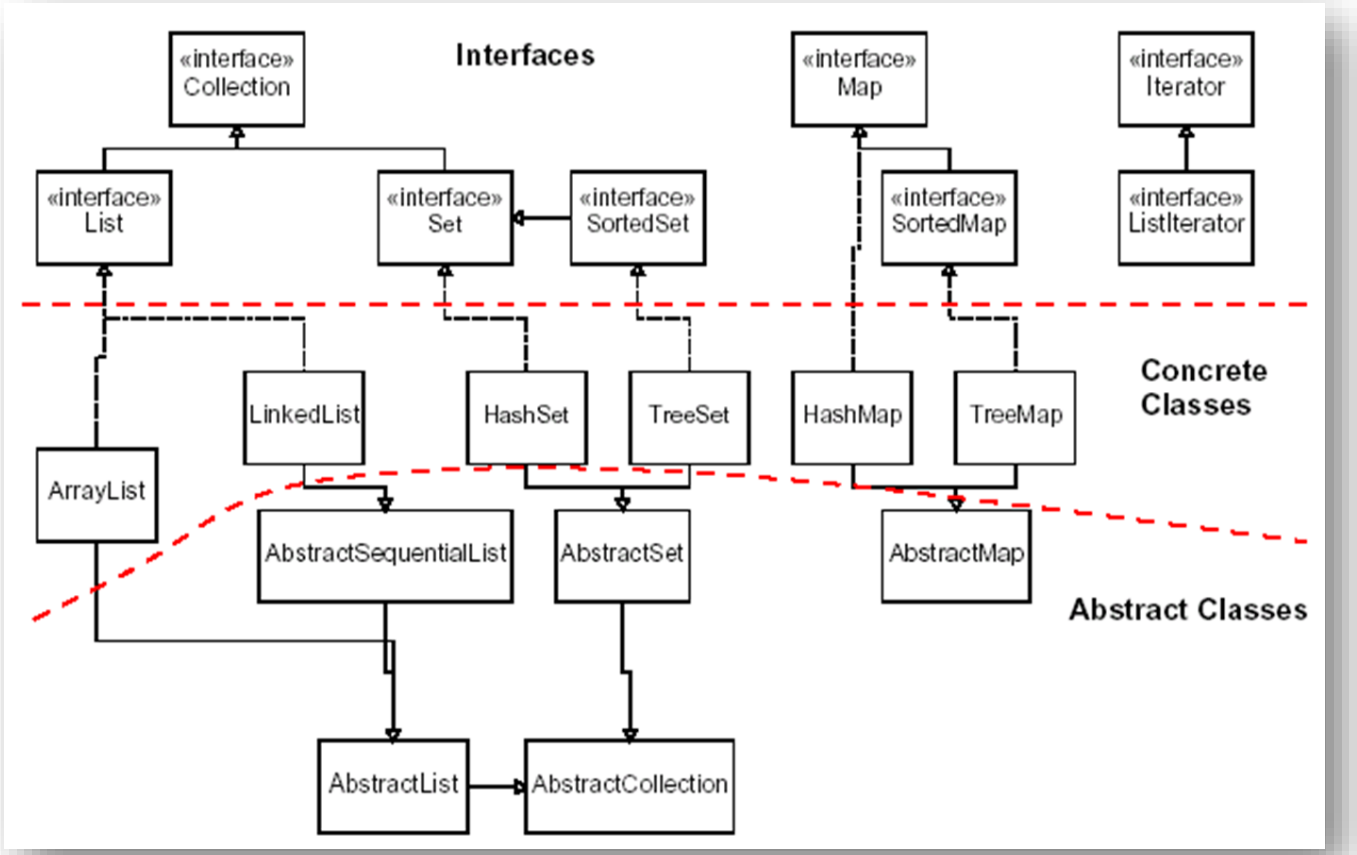
Collection è la radice della gerarchia delle collection (https://docs.oracle.com/javase/9/docs/api/java/util/Collection.html)

- rappresenta gruppi di oggetti
  - gli oggetti (elementi) possono essere o non essere duplicati
  - gli oggetti possono essere o non essere ordinati

esistono implementazioni concrete di sottointerfacce

- (List, Set)

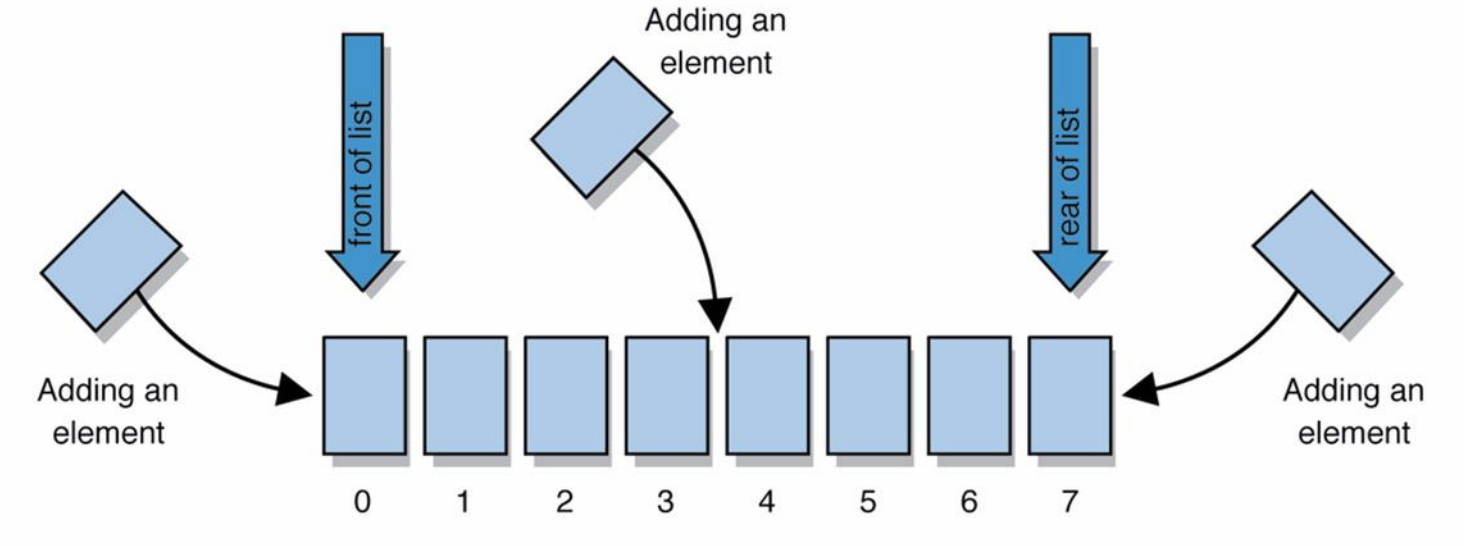
Java Collection Framework



List

List definisce il concetto di lista ordinata (o sequenza)

- insieme di elementi posti in un certo ordine
- ogni elemento è accessibile attraverso un indice (0-based index)
- gli elementi possono essere inseriti in testa, in coda o in qualsiasi altra posizione
- implementazioni: ArrayList, LinkedList e Vector



ArrayList - LinkedList

ArrayList

- ottimizzato l'accesso casuale (basato su array)
- non ottimizzati l'inserimento e l'eliminazione all'interno della lista

LinkedList

- ottimizzato l'accesso sequenziale, per l'inserimento e l'eliminazione
- indicato per implementare pile (LIFO) e code (FIFO)
- contiene i metodi:
  - `addFirst(), addLast(), getFirst(),`
  - `getLast(), removeFirst( ), removeLast()`

`ArrayList` methods

`add(`**value**`)`	appends value at end of list
`add(`**index**`, `**value**`)`	inserts given value just before the given index, shifting subsequent values to the right
`clear()`	removes all elements of the list
`indexOf(`**value**`)`	returns first index where given value is found in list (-1 if not found)
`get(`**index**`)`	returns the value at given index
`remove(`**index**`)`	removes/returns value at given index, shifting subsequent values to the left
`set(`**index**`, `**value**`)`	replaces value at given index with given value
`size()`	returns the number of elements in list
`toString()`	returns a string representation of the list such as `"[3, 42, -7, 15]"`

`ArrayList` methods

`addAll(`**list**`)` `addAll(`**index**`, `**list**`)`	adds all elements from the given list to this list (at the end of the list, or inserts them at the given index)
`contains(`**value**`)`	returns true if given value is found somewhere in this list
`containsAll(`**list**`)`	returns true if this list contains every element from given list
`equals(`**list**`)`	returns true if given other list contains the same elements
`iterator()` `listIterator()`	returns an object used to examine the contents of the list (seen later)
`lastIndexOf(`**value**`)`	returns last index value is found in list (-1 if not found)
`remove(`**value**`)`	finds and removes the given value from this list
`removeAll(`**list**`)`	removes any elements found in the given list from this list
`retainAll(`**list**`)`	removes any elements *not* found in given list from this list
`subList(`**from**`, `**to**`)`	returns the sub-portion of the list between indexes **from** (inclusive) and **to** (exclusive)
`toArray()`	returns the elements in this list as an array

ArrayList vs Array

ARRAY

costruttori

- String[] names = new String[5];

inserimento valori

- names[0] = "Jessica";

accesso ai valori

- String s = names[0];

ARRAYLIST

costruttori

- ArrayList<String> list = new ArrayList<String>();

inserimento valori

- list.add("Jessica");

accesso ai valori

- String s = list.get(0);

ArrayList vs Array

ARRAY

ricercare i valori che iniziano per "It"

```
for (int i = 0; i < names.length; i++) {
    if (names[i].startsWith("It")) { ... }
}
```

ricercare se è presente il valore "Italia"

```
for (int i = 0; i < names.length; i++) {
    if (names[i].equals("Italia")) { ... }
}
```

ARRAYLIST

ricercare i valori che iniziano per "It"

```
for (int i = 0; i < list.size(); i++) {
    if (list.get(i).startsWith("It")) { ... }
}
```

ricercare se è presente il valore "Italia"

```
if (list.contains("Italia")) { ... }
```

ArrayList e tipi primitivi

il tipo degli elementi di un `ArrayList` deve essere un object type

- non può essere un tipo primitivo

```
// illegal -- int cannot be a type parameter
ArrayList<int> list = new ArrayList<int>();
```

possiamo utilizzare le classi *wrapper*

```
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

Classi wrapper

Primitive Type	Wrapper Type
int	Integer
double	Double
char	Character
boolean	Boolean

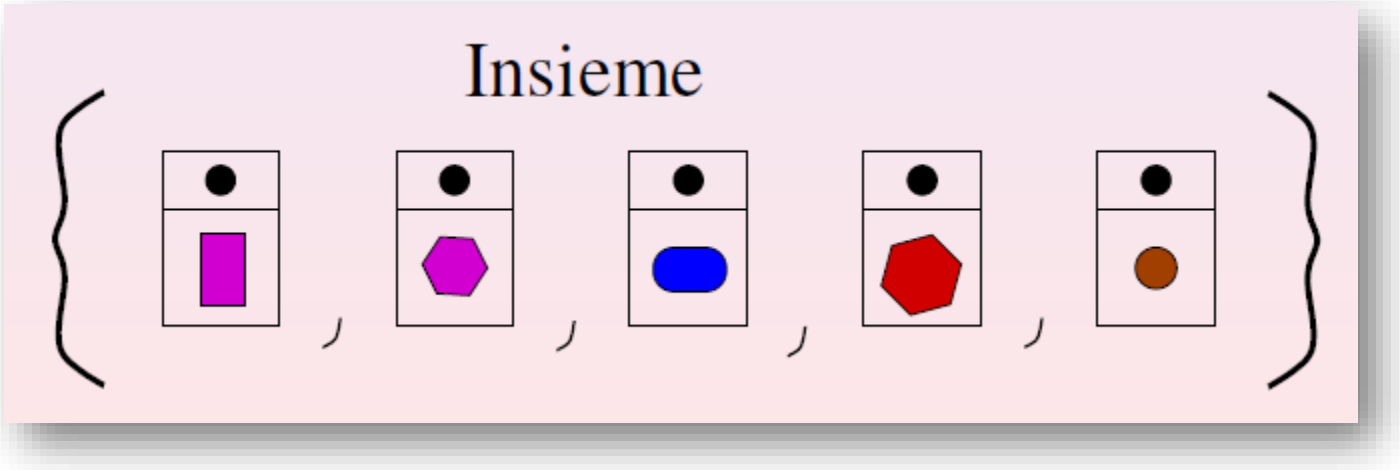
```
ArrayList<Double> voti = new ArrayList<Double>();
voti.add(7.5);
voti.add(4.5);
...
double mioVoto = voti.get(0);
```

Set

Set definisce il concetto di insieme

- gruppo di elementi non duplicati
  - (non contiene e1 e e2 se e1.equals(e2).

implementazioni: HashSet



Classi «generiche»

Le collezioni sono un esempio di classi «parametrizzate» (generic classes)

Quando utilizziamo una collezione dobbiamo specificare due tipi:

- Il tipo della collezione
- Il tipo degli elementi

Esempio:

- `ArrayList<Persona>`
- `ArrayList<Cerchio>`
- `ArrayList<String>`

ArrayList è una classe della libreria java.util che fornisce una semplice implementazione di un raggruppamento non ordinato di oggetti (https://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html)

Implementa le funzionalità di una lista, ha i metodi:

- add
- get
- size
- …

Esempio: GestoreMusica

La classe GestoreMusica gestisce semplicemente i nomi dei file dei brani

Non gestisce informazioni relative al titolo, all'artista, alla durata ecc.

Contiene un ArrayList di stringhe che rappresentano i nomi dei file

Delega

- La classe delega la responsabilità della gestione delle operazioni alla collezione

GestoreMusica: Parte del codice

```
import java.util.ArrayList;

public class GestoreMusica {
   // ArrayList per memorizzare i nomi dei file dei brani musicali
   private ArrayList<String> brani;

   public GestoreMusica () {
      brani = new ArrayList<String>();
   }

...
```

Alcuni metodi

```
/**
 * aggiunge un brano alla collezione
 * @param nomeFile il brano da aggiungere
 */
public void aggiungiBrano(String nomeFile)
{
    brani.add(nomeFile);
}

/**
 * Numero di brani presenti nella collezione
 * @return il numero di brani della collezione
 */
public int getNumeroBrani()
{
    return brani.size();
}
```

```
/**
     * Visualizza un brano
     * @param indice indice del brano
     */
    public void visualizzaBrano(int indice)
    {
        if(indice >= 0 && indice < brani.size()) {
            String nomeFile = brani.get(indice);
            System.out.println(nomeFile);
        }
    }

    /**
     * Elimina un brano dalla collezione
     * @param indice indice del brano
     */
    public void eliminaBrano(int indice)
    {
        if(indice >= 0 && indice < brani.size()) {
            brani.remove(indice);
        }
    }
```

Indice degli elementi

Il primo elemento aggiunto alla collezione ha indice 0, il secondo indice 1 …

Il metodo get(indice) permette di accedere direttamente ad un elemento della collezione

L'utilizzo di un indice errato genera un messaggio di errore (indiceOutOfBoundsException)

Il metodo remove(indice) elimina un elemento dalla collezione

- La rimozione causa la modifica degli indici degli altri elementi della collezione

Oltre che come ultimo è possibile inserire un elemento in una posizione specifica

Accedere a tutti gli elementi di una collezione

Il ciclo foreach permette di accedere sequenzialmente a tutti gli elementi di una collezione

```
for(<tipoElemento> elemento : <collezione>) {
  <corpo del ciclo>
}
```

Esempio: visualizza tutti I brani

```
public void visualizzaBrani() {
  for(String nomeBrano : brani) {
    System.out.println(nomeBrano);
  }
}
```

Esercizio

Aggiungere alla classe GestoreMusica il metodo void cerca(String stringaRicerca) che visualizza tutti i brani che contengono stringaRicerca

- Utilizzare il metodo java.lang.String.contains()
- Se non si trova nessun brano visualizzare un messaggio di errore

Aggiungere il metodo void visualizzaTutti() che visualizza tutti I brani

Aggiungere il metodo void visualizzaPrimo(String stringaRicerca) che visualizza il primo brano che contiene la stringa di ricerca

- for-each o while?
- while(boolean condition) {
  loop body
  }

Ricerca in una collezione

La ricerca può aver successo dopo un indefinito numero di iterazioni

La ricerca fallisce dopo aver esaurito ogni possibilità

```
int indice = 0;
boolean trovato = false;
while(indice < miaColl.size() && !trovato) {
  elemento = miaColl.get(indice);
  if (elemento ...) {
    trovato = true;
    ...
  }
  indice++;
}
```

GestoreMusica (v2)

Si vogliono memorizzare più informazioni per ogni brano:

- Artista
- Titolo
- Nome del file

Realizzare la classe Brano che permette di gestire queste informazioni

- Attributi
- Costruttori
- Setter e getter
- Metodi
  - String getInformazioni()
    - Restituisce una stringa formata da Artista + Titolo + Nome del file

GestoreMusica (v2)

Modificare la classe GestoreMusica in modo che l'arrayList contenga i Brani e non più stringhe

Inserire il metodo visualizzaBrani() che visualizza tutti i brani presenti nella collezione

```
public void visualizzaBrani() {
   for(Brano brano : brani) {
     System.out.println(brano.getInformazioni());
   }
}
```

Questo è un esempio di responsibility-driven design (si delega alla classe la sua gestione)

iteratori

Un iteratore è un oggetto che fornisce le funzionalità per iterare su tutti gli elementi di una collezione

Il metodo iterator() di ogni collezione restituisce un oggetto iteratore

```
Iterator<ElementType> it = myCollection.iterator();

while(it.hasNext()) {
  // utilizzare it.next() per ottenere l'elemento successivo
  // utilizzare questo elemento
}
```

Esempio iteratore

```
import java.util.ArrayList;
import java.util.Iterator;
...
public void visualizzaBrani() {
Iterator<Brano> it = brani.iterator();
while(it.hasNext()) {
        Brano b = it.next();
        System.out.println(b.getInformazioni());
    }
}
```

Rimozione di un elemento

Per cercare e quindi rimuovere un elemento non è possibile utilizzare un cicle for-each

Si otterrebbe il seguente messaggio di errore: ConcurrentModificationException

```
Iterator<Brano> it = brani.iterator();
while(it.hasNext()) {
    Brano b = it.next();
    String artista = b.getArtista();
    if(artista.equals(artistaDaEliminare)) {
        it.remove();
    }
}
```

Utilizzare il metodo remove() dell'iteratore e non quello della collezione!

Esercizi

Implementare il metodo void rimuoviTitolo(String titoloDaRimuovere) che elimina dalla collezione tutti i brani con il titolo specificato

Implementare il metodo void cambiaTitolo(String vecchioTitolo, String nuovoTitolo) che rinomina tutti i brani con vecchioTitolo in nuovoTitolo