collezioni

Java

- 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

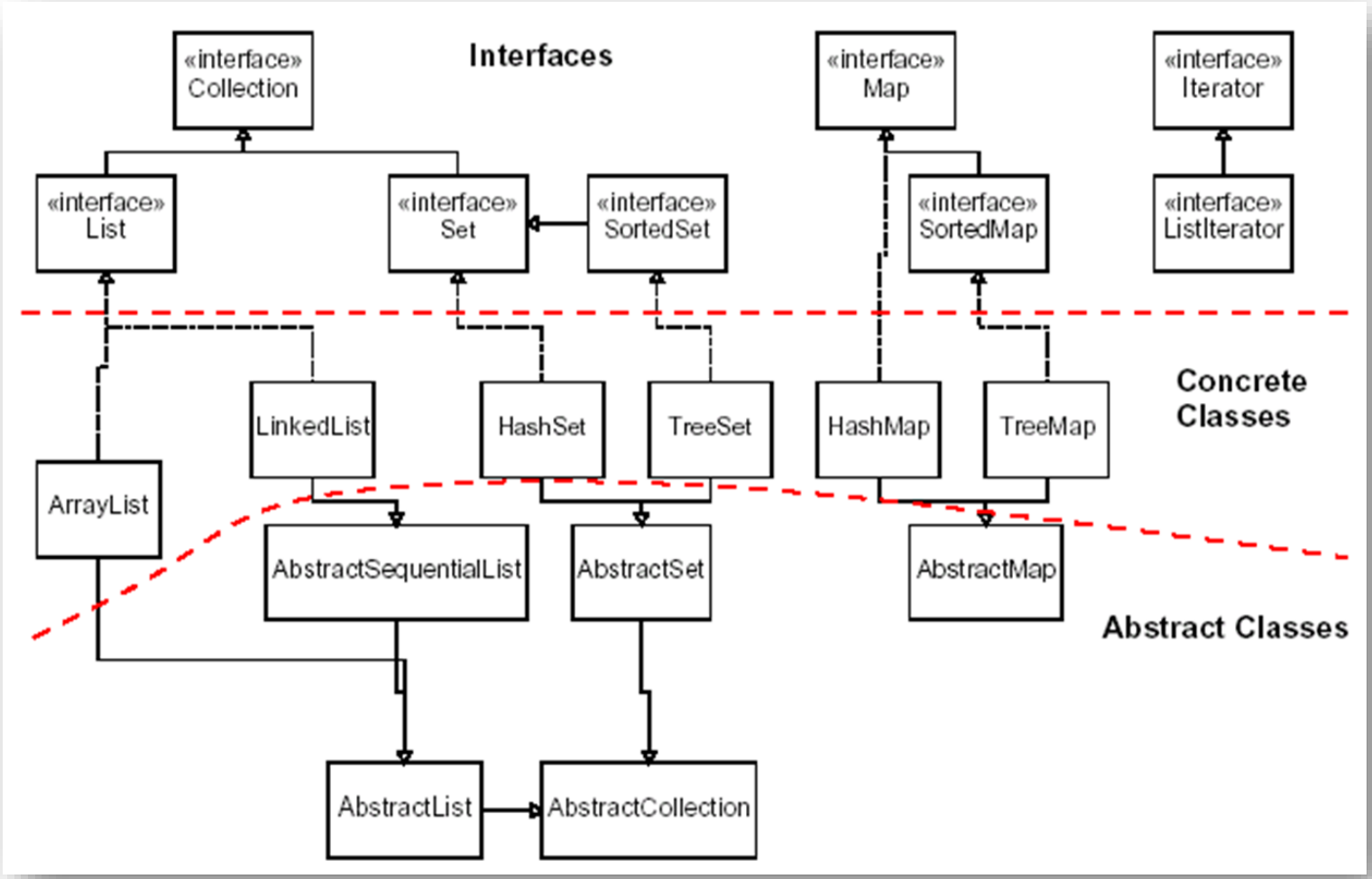
- una delle caratteristiche dei linguaggi object oriented che li rende molto potenti è la presenza di ***librerie di classi***
- le librerie tipicamente contengono decine o centinaia di classi utili per gli sviluppatori e utilizzabili in un ampio insieme di applicazioni
- in Java le librerie vengono definite ***packages***
- il package ***java.util*** contiene classi per la gestione di collezioni di oggetti

caratteristiche comuni alle collezioni

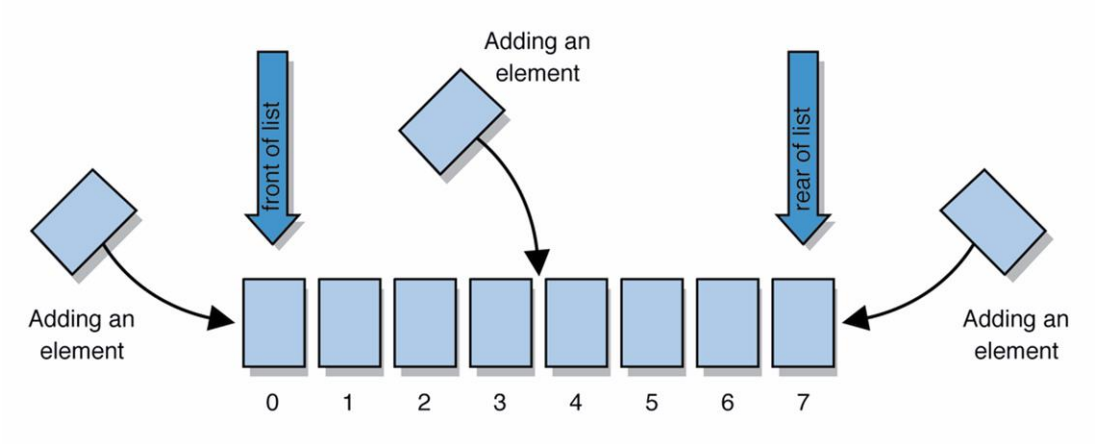
- possibilità di aumentare la ***capacità*** (se necessario)
- mantenere un contatore privato del ***numero di oggetti*** presenti
  - il valore è accessibile mediante il metodo size()
- mantenere l'***ordine*** di inserimento degli elementi
- ogni elemento ha un ***indice***
  - il valore dell'indice 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?

Java Collection Framework

A. Ferrari



- ***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***
  - 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

A. Ferrari

`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

A. Ferrari

`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 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*

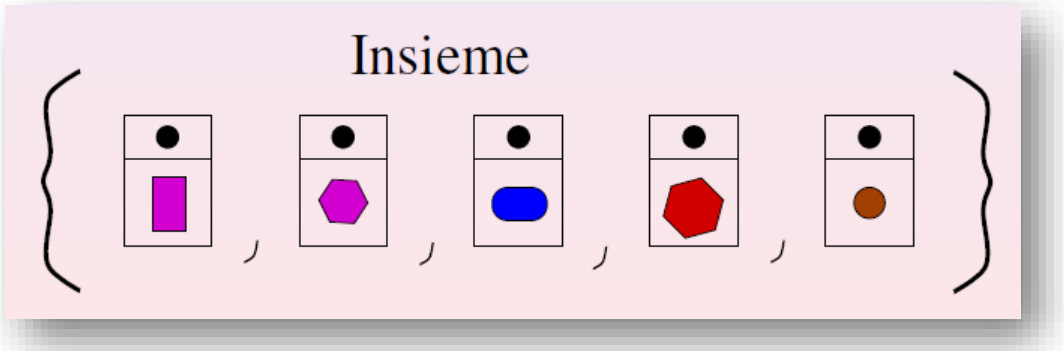
```
// creates a list of ints
ArrayList<Integer> list = new ArrayList<Integer>();
```

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 definisce il concetto di insieme
  - gruppo di elementi non duplicati
    - (non contiene e1 e e2 se e1.equals(e2).
- implementazioni: HashSet



- 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

```
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>();
    }

  ...
```

```
/** 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 ***(indexOutOfBoundsException)***
- 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

una versione del ciclo for 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);
  }
}
```

- 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
}
```

- 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++;
}
```

- 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

- 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)

- 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());
    }
}
```

- 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!***

- 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