Metodologie di Programmazione

Lezione 22: Le collezioni (parte 1)

Lezione 22: Sommario

UNITELMA SAPIENZA

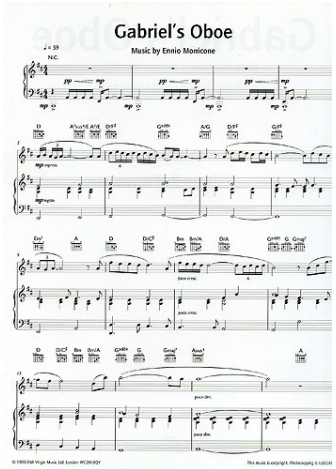
SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

- Strutture dati in Java: le collection
- Gerarchia delle Collection
- Iterator
- ArrayList e LinkedList
- ListIterator
- Insiemi

A cosa serve una struttura dati?

- A memorizzare e organizzare i dati in memoria così da utilizzarli in modo efficiente

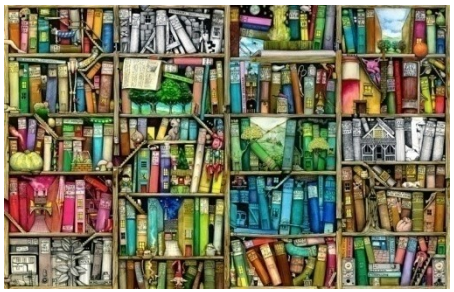
Una mano di carte



Uno spartito

Una sequenza di canzoni

Una squadra di pallavolo

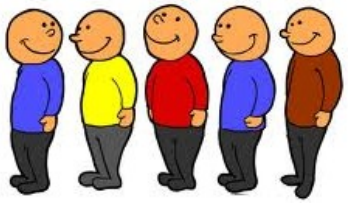


Una libreria

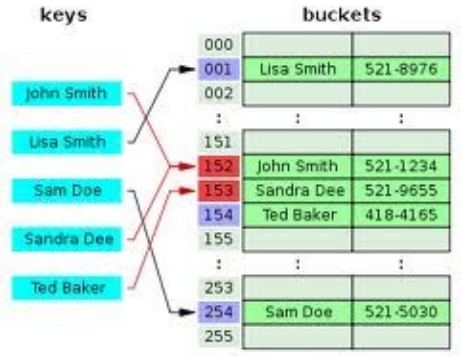
Caratteristiche di una struttura dati

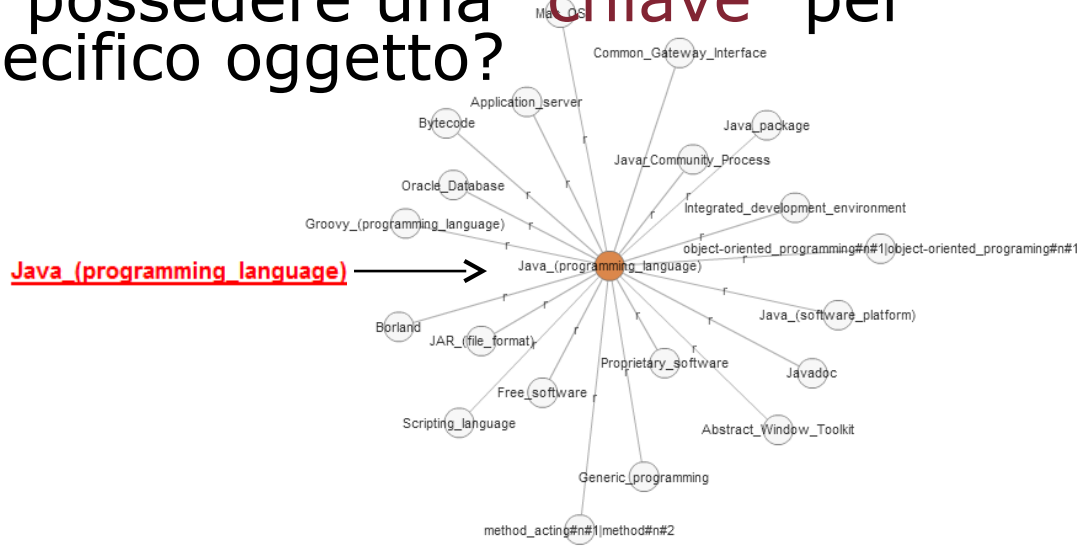
- E' necessario mantenere un ordine?

- Gli oggetti nella struttura possono ripetersi?



- E' utile/necessario possedere una "chiave" per accedere a uno specifico oggetto?



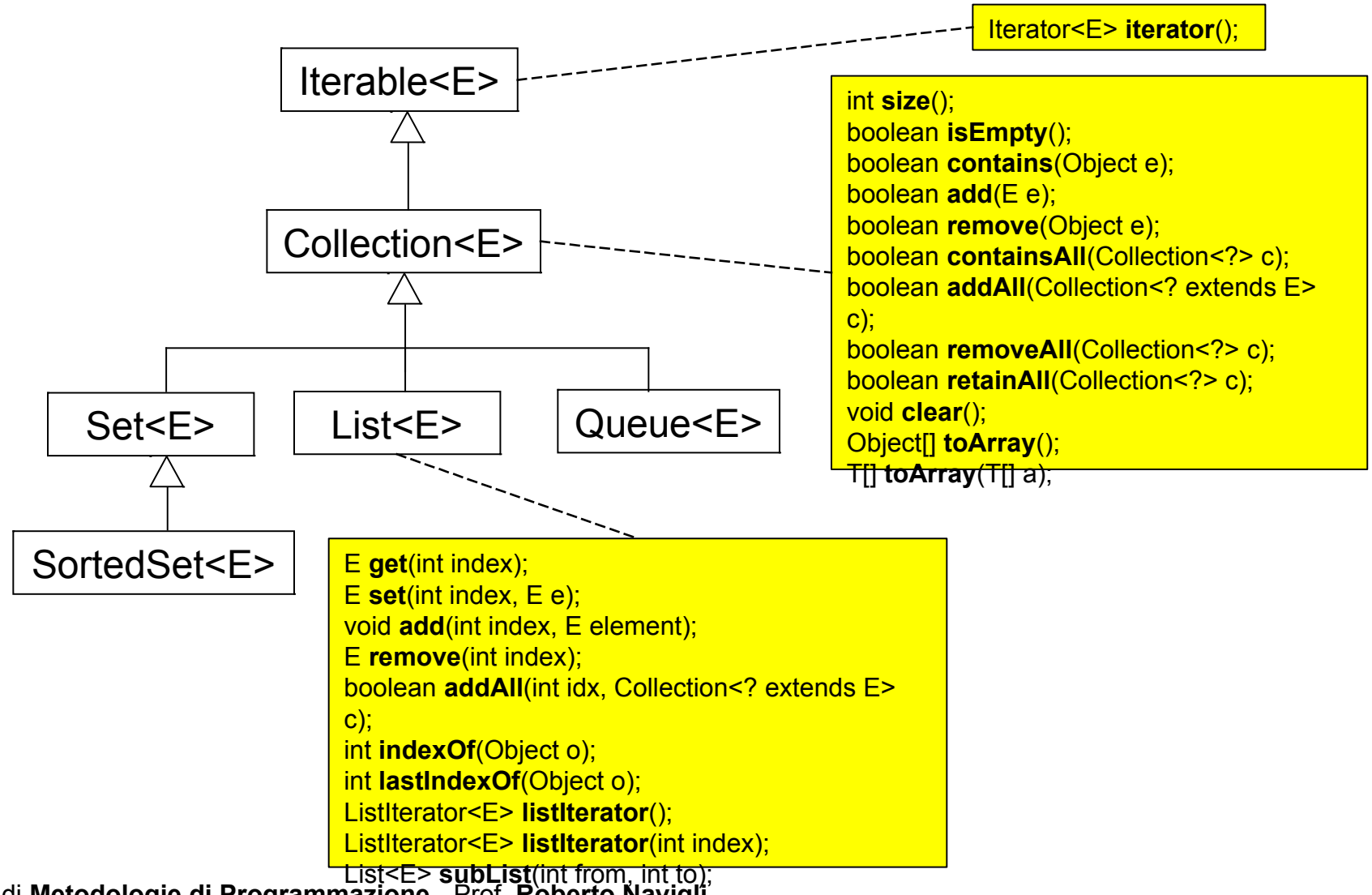


Le Collection

- Le collezioni in Java sono rese disponibili mediante il framework delle collezioni (Java Collection Framework)
- Strutture dati già pronte all'uso
  - con interfacce e algoritmi per manipolarle
- Contengono e "strutturano" **riferimenti** ad altri oggetti
  - Tipicamente tutti "dello stesso tipo"
- Alcune interfacce del framework:

Interfaccia	Descrizione
Collection	L'interfaccia alla radice della gerarchia di collezioni
Set	Una collezione senza duplicati
List	Una collezione ordinata che può contenere duplicati
Map	Associa coppie di (chiave, valore), senza chiavi duplicate
Queue	Una collezione first-in, first-out che modella una coda

Gerarchia delle interfacce di tipo Collection



Iterazione su una collezione

1. Mediante gli Iterator:

```
// "vecchio" stile, ma maggiore controllo
Iterator<Integer> i = collezione.iterator();
while(i.hasNext())
{
    int k = i.next();
    System.out.println(k);
}
```

Finché c'è ancora un elemento

Ottieni il prossimo elemento

2. Mediante il costrutto "for each":

```
// "nuovo" stile: for each, più elegante
for (Integer k : collezione)
    System.out.println(k);
```

Per ogni elemento k nella collezione

3.

```
// su collezioni che implementano List: elemento per elemento
for (int j = 0; j < collezione.size(); j++)
{
    int k = collezione.get(j);
    System.out.println(k);
}
```

Accesso elemento per elemento

L'interfaccia java.util.Iterator<E>

- E' un'interfaccia fondamentale che permette di **iterare** su collezioni
- Espone tre metodi:

Metodo	**Descrizione**
boolean **hasNext**()	Restituisce true se esiste ancora un successivo elemento nella collezione
E **next**()	Restituisce l'elemento successivo
void **remove**()	Rimuove l'elemento corrente (operazione **opzionale**)

- E' in relazione con l'interfaccia Iterable nel senso che chi implementa Iterable restituisce un Iterator sull'oggetto-collezione

Esercizio: il mio array iterabile

- Scrivere una classe che implementa un array di interi iterabile

- Utilizzare l'interfaccia Iterable<Integer> e le classi interne

Esempio: Il Jukebox


```java
import java.util.ArrayList;
import java.util.Iterator;

/**
 * Rappresenta un JukeBox di canzoni su cui si può iterare
 *
 * @author navigli
 */
public class Jukebox implements Iterable<Canzone>
{
    /**
     * Elenco delle canzoni
     */
    private ArrayList<Canzone> canzoni = new ArrayList<Canzone>();

    /**
     * Permette di aggiungere una canzone
     * @param c la canzone da aggiungere
     */
    public void addCanzone(Canzone c)
    {
        canzoni.add(c);
    }

    @Override
    public Iterator<Canzone> iterator()
    {
        return canzoni.iterator();
    }
}
```

Collezioni fondamentali

- Saranno molto utili nel tempo:
  - ArrayList
  - LinkedList
    - Estendono entrambe AbstractList
  - HashSet
  - TreeSet
    - Estendono entrambe AbstractSet
  - HashMap
  - TreeMap
    - Estendono AbstractMap
  - LinkedHashMap
    - Estende HashMap

Le liste: ArrayList e LinkedList

- Basate su List, una sottointerfaccia di Collection e di Iterable
- Le classi liste sono: ArrayList e LinkedList
  - estendono AbstractList e implementano l'interfaccia List
- ArrayList implementa la lista mediante un array (eventualmente ridimensionato)
  - Capacità iniziale: 10 elementi
- LinkedList implementa la lista mediante elementi linkati

Metodi di ArrayList

Method Summary	
boolean	`add(E e)` Appends the specified element to the end of this list.
void	`add(int index, E element)` Inserts the specified element at the specified position in this list.
boolean	`addAll(Collection<? extends E> c)` Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's Iterator.
boolean	`addAll(int index, Collection<? extends E> c)` Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	`clear()` Removes all of the elements from this list.
Object	`clone()` Returns a shallow copy of this `ArrayList` instance.
boolean	`contains(Object o)` Returns `true` if this list contains the specified element.
void	`ensureCapacity(int minCapacity)` Increases the capacity of this `ArrayList` instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
E	`get(int index)` Returns the element at the specified position in this list.
int	`indexOf(Object o)` Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
boolean	`isEmpty()` Returns `true` if this list contains no elements.
int	`lastIndexOf(Object o)` Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
E	`remove(int index)` Removes the element at the specified position in this list.
boolean	`remove(Object o)` Removes the first occurrence of the specified element from this list, if it is present.
protected void	`removeRange(int fromIndex, int toIndex)` Removes from this list all of the elements whose index is between `fromIndex`, inclusive, and `toIndex`, exclusive.
E	`set(int index, E element)` Replaces the element at the specified position in this list with the specified element.
int	`size()` Returns the number of elements in this list.
Object[]	`toArray()` Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	`toArray(T[] a)` Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.
void	`trimToSize()` Trims the capacity of this `ArrayList` instance to be the list's current size.

Alcuni metodi aggiuntivi di LinkedList

Metodo	Descrizione
void **addFirst**(E e)	Aggiunge l'elemento in testa alla lista
void **addLast**(E e)	Aggiunge l'elemento in coda alla lista
Iterator<E> **descendingIterator**()	Restituisce un iteratore che parte dall'ultimo elemento della lista e si sposta verso sinistra
E **getFirst**()	Restituisce il primo elemento della lista
E **getLast**()	Restituisce l'ultimo elemento della lista
E **removeFirst**()	Rimuove e restituisce il primo elemento
E **removeLast**()	Rimuove e restituisce l'ultimo elemento
E **pop**()	Elimina e restituisce l'elemento in cima alla lista vista come pila
void **push**(E e)	Inserisce un elemento in cima alla lista vista come pila

Iterare sulle liste in entrambe le direzioni

- listIterator() restituisce un iteratore bidirezionale (interfaccia ListIterator che estende Iterator) per la lista
- Da sinistra verso destra:

```
ListIterator<Integer> i = l.listIterator();
while(i.hasNext())
    System.out.println(i.next());
```

- Da destra verso sinistra (se si specifica un intero, si parte da quella posizione: es. list.listIterator(list.size()))

```
ListIterator<Integer> i = l.listIterator(l.size());
while(i.hasPrevious())
    System.out.println(i.previous());
```

ListIterator

Method Summary	
void	add(E e) Inserts the specified element into the list (optional operation).
boolean	hasNext() Returns `true` if this list iterator has more elements when traversing the list in the forward direction.
boolean	hasPrevious() Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
E	next() Returns the next element in the list.
int	nextIndex() Returns the index of the element that would be returned by a subsequent call to `next`.
E	previous() Returns the previous element in the list.
int	previousIndex() Returns the index of the element that would be returned by a subsequent call to `previous`.
void	remove() Removes from the list the last element that was returned by `next` or `previous` (optional operation).
void	set(E e) Replaces the last element returned by `next` or `previous` with the specified element (optional operation).

Insiemi: HashSet e TreeSet

- Basati su Set, una sottointerfaccia di Collection e di Iterable
- Gli insiemi sono Collection che contengono elementi **tutti distinti**
- Le classi insiemi, HashSet e TreeSet, estendono AbstractSet e implementano l'interfaccia Set
- HashSet memorizza gli elementi in una tabella di hash
- TreeSet memorizza gli elementi in un albero mantenendo un ordine sugli elementi

HashSet: Un esempio

- Salviamo nomi e cognomi in due insiemi:

```
HashSet<String> nomi = new HashSet<String>();
HashSet<String> cognomi = new HashSet<String>();

nomi.add("mario");
cognomi.add("rossi");

nomi.add("mario");
cognomi.add("bianchi");

nomi.add("mario");
cognomi.add("verdi");

nomi.add("luigi");
cognomi.add("rossi");

nomi.add("luigi");
cognomi.add("bianchi");

System.out.println(nomi);
System.out.println(cognomi);
```

- ttieniamo:

```
[mario, luigi]
[verdi, bianchi, rossi]
```

TreeSet: Un esempio

- Salviamo nomi e cognomi in due insiemi:

```
TreeSet<String> nomi = new TreeSet<String>();
TreeSet<String> cognomi = new TreeSet<String>();

nomi.add("mario");
cognomi.add("rossi");

nomi.add("mario");
cognomi.add("bianchi");

nomi.add("mario");
cognomi.add("verdi");

nomi.add("luigi");
cognomi.add("rossi");

nomi.add("luigi");
cognomi.add("bianchi");

System.out.println(nomi);
System.out.println(cognomi);
```

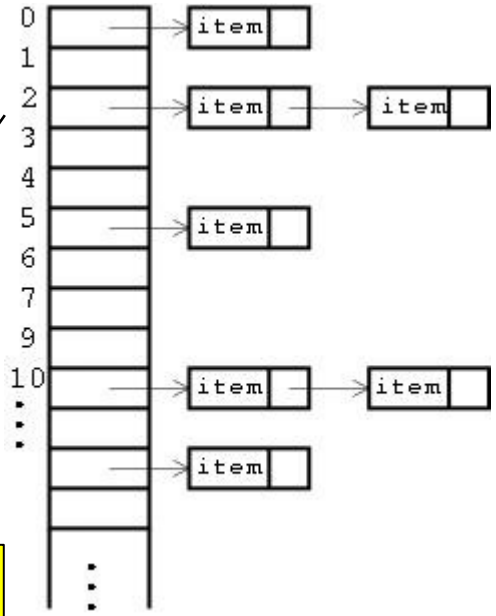
- Stampandoli, otteniamo:

```
[luigi, mario]
[bianchi, rossi, verdi]
```

Ordinati secondo l'ordinamento naturale del tipo (String)

Come funziona un HashSet?

- Si fonda sul concetto di tabella hash



Questo numero è una funzione dell'hashCode() dell'oggetto usato come chiave