

Java SE: Collezioni, ...

- Inner class
- Generic
- Iterator
- Collezioni
- Reflection
- Multithreading
- Progetto di riferimento
 - <https://github.com/egalli64/mpjp> (*modulo 4*)

Inner class

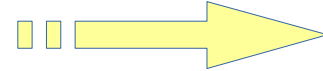
- Nested class: classe definita all'interno di un'altra classe
- La nested class ha accesso diretto ai membri della classe in cui è definita
- È possibile definirla come locale ad un blocco
- Inner class: non-static nested class
- Utili (ad es.) per semplificare la gestione di eventi

Generic

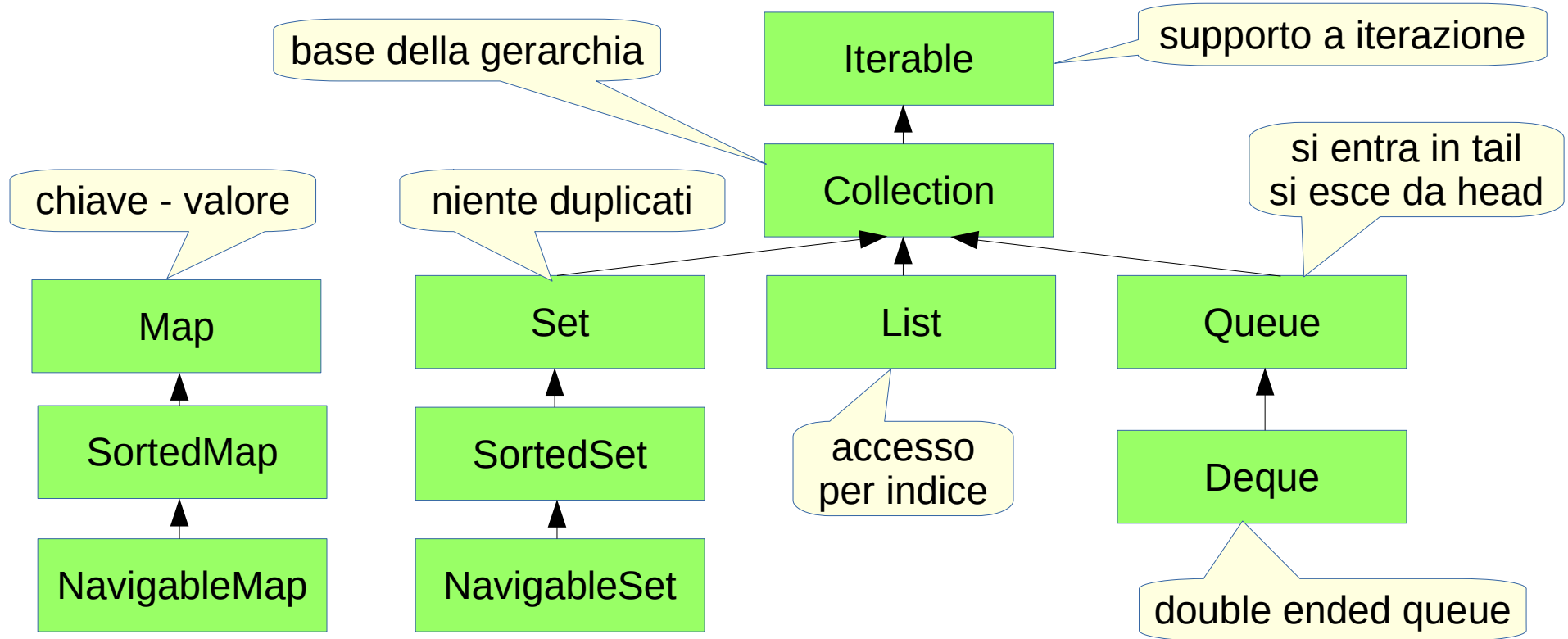
- Supporto ad algoritmi generici che operano allo stesso modo su tipi differenti (es: collezioni)
- Migliora la type safety del codice
 - Errori possono essere intercettati in fase di compilazione
- In Java è implementato solo per reference types
- Il tipo (o tipi) utilizzato dal generic è indicato tra parentesi angolari (minore, maggiore)
 - Esempio: `ArrayList<String>` è un'ArrayList di stringhe

Java Collections Framework

- Gestione di gruppi di oggetti (**solo reference!**)
- Efficienza, performance, interoperabilità, estensibilità, adattabilità
- Basate su alcune interfacce standard
- Tutte le collezioni implementano Iterable, e dunque definiscono
 - **iterator()** che ritorna un **Iterator** (da Java 8 anche spliterator())
 - Loop idiomatico sulla collezione “*while has next*” + supporto a for-each
 - forEach() che permette di eseguire un’azione su ogni elemento (Java 8)
- La classe **Collections** contiene algoritmi generici
 - min(), max(), sort(), ...



Interfacce per Collection



Alcuni metodi in Collection<E>

- boolean add(E)
- boolean addAll(Collection<? extends E>)
- void clear()
- boolean contains(Object);
- boolean equals(Object);
- boolean isEmpty();
- Iterator<E> iterator();
- boolean remove(Object);
- int size();
- Object[] toArray();
- <T> T[] toArray(T[]);

Alcuni metodi in List<E>

- void add(int, E)
 - Overload di add(), per inserire nella posizione indicata
- E get(int)
- int indexOf(Object)
 - Indice dell'elemento indicato – se non lo trova: -1
- E remove(int)
- E set(int, E)
 - Combina remove + add

Alcuni metodi in SortedSet<E>

- E first()
- E last()
- SortedSet<E> subSet(E, E)
 - Intervallo chiuso a sinistra, aperto a destra

Alcuni metodi in NavigableSet<E>

- E ceiling(E), E floor(E)
 - Elemento pari o maggiore/minore – o null
- E higher(E), E lower(E)
 - Elemento strettamente maggiore/minore – o null
- E pollFirst(), E pollLast()
 - Legge e rimuove primo/ultimo elemento – o null
- Iterator<E> descendingIterator()
- NavigableSet<E> descendingSet()

Alcuni metodi in Queue<E>

- `boolean offer(E e)`
 - Aggiunge un elemento (a destra) – o ritorna `false`
- `E element()`
 - Legge un elemento (a sinistra) – o `NoSuchElementException`
- `E peek()`
 - Legge un elemento (a sinistra) – o `null`
- `E remove()`
 - Rimuove un elemento (a sinistra) e lo ritorna – o `NoSuchElementException`
- `E poll()`
 - Rimuove un elemento (a sinistra) e lo ritorna – o `null`

Alcuni metodi in Deque<E>

- void addFirst(E), void addLast(E)
- boolean offerFirst(E), boolean offerLast(E)
 - Aggiungono un elemento a sinistra/destra – o IllegalStateException (add) / false (offer)
- E getFirst(), E getLast()
- E peekFirst(), E peekLast()
 - Leggono un elemento a sinistra/destra – o NoSuchElementException (get) / null (peek)
- E removeFirst(), E removeLast()
- E pollFirst(), E pollLast()
 - Leggono e rimuovono un elemento a sinistra/destra – o NoSuchElementException (remove) / null (poll)
- E pop(), void push(E)
 - Equivalenti a removeFirst() e addFirst(), il loro nome indica che il Deque è usato come uno Stack

Alcuni metodi in Map<K, V>

Map.Entry<K,V>

- K getKey(), V getValue(), V setValue(V)

- Set<Map.Entry<K, V>> entrySet()
 - Tutte le coppie chiave/valore nella mappa
- Set<K> keySet()
 - Tutte le chiavi nella mappa
- Collection<V> values()
 - Tutti i valori nella mappa
- boolean containsKey(Object)
- boolean containsValue(Object)
- V get(Object) // or null
- V getOrDefault(Object, V)
- V put(K, V)
 - Aggiunge K-V o cambia V → null o prev val
- V putIfAbsent(K, V)
 - Se manca K, aggiunge K-V → null o cur val
- V remove(Object)
 - Elimina K → prev val o null
- boolean remove(Object, Object)
 - Elimina K-V o → false
- V replace(K key, V value)

Metodi in NavigableMap<K, V>

- Map.Entry<K,V> ceilingEntry(K)
 - Entry con key maggiore o uguale, o null
- Map.Entry<K,V> higherEntry(K)
 - Entry con key strettamente maggiore, o null
- Map.Entry<K,V> floorEntry(K)
 - Entry con key minore o uguale, o null
- Map.Entry<K,V> lowerEntry(K)
 - Entry con key strettamente minore, o null
- K ceilingKey(K)
- K higherKey(K key)
- K floorKey(K)
- K lowerKey(K)
- Map.Entry<K,V> firstEntry()
- Map.Entry<K,V> lastEntry()
- NavigableSet<K> navigableKeySet()
- Map.Entry<K,V> pollFirstEntry()
- Map.Entry<K,V> pollLastEntry()
 - Rimuove l'entry e la ritorna, o null
- NavigableMap<K,V> headMap(K, boolean)
- NavigableMap<K,V> tailMap(K, boolean)
 - Submap con limite incluso o meno
- SortedMap<K,V> subMap(K, K)
 - Submap aperta a destra

ArrayList<E>

- implements List<E>
- Array dinamico vs standard array (dimensione fissa)
- Ctors
 - ArrayList() // capacity = 10
 - ArrayList(int) // set capacity
 - ArrayList(Collection<? extends E>) // copy

LinkedList<E>

- implements List<E>, Deque<E>
- Lista doppiamente linkata
- Accesso diretto solo a head e tail
- Ctors
 - LinkedList() // vuota
 - LinkedList(Collection<? extends E>) // copy

HashSet<E>

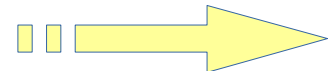
- implements Set<E>
- Basata sull'ADT hash table, $O(1)$, nessun ordine
- Ctors:
 - HashSet() // vuota, capacity 16, load factor .75
 - HashSet(int) // capacity
 - HashSet(int, float) // capacity e load factor
 - HashSet(Collection<? extends E>) // copy

LinkedHashSet<E>

- extends HashSet<E>
- Permette di accedere ai suoi elementi in ordine di inserimento
- Ctors:
 - `LinkedHashSet()` // capacity 16, load factor .75
 - `LinkedHashSet(int)` // capacity
 - `LinkedHashSet(int, float)` // capacity, load factor
 - `LinkedHashSet(Collection<? extends E>)` // copy

TreeSet<E>

- implements NavigableSet<E>
- Basata sull'ADT albero → ordine, $O(\log(N))$
- Gli elementi inseriti devono implementare Comparable ed essere tutti mutualmente comparabili
- Ctors:
 - TreeSet() // vuoto, ordine naturale
 - TreeSet(Collection<? extends E>) // copy
 - TreeSet(Comparator<? super E>) // sort by comparator
 - TreeSet(SortedSet<E>) // copy + comparator



TreeSet e Comparator

ordine naturale

comparator

plain

reversed

Java 8 lambda

```
List<String> data = Arrays.asList("alpha", "beta", "gamma", "delta");

TreeSet<String> ts = new TreeSet<>(data);

class MyStringComparator implements Comparator<String> {
    public int compare(String s, String t) {
        return s.compareTo(t);
    }
}

MyStringComparator msc = new MyStringComparator();

TreeSet<String> ts2 = new TreeSet<>(msc);
ts2.addAll(data);

TreeSet<String> ts3 = new TreeSet<>(msc.reversed());
ts3.addAll(data);

TreeSet<String> ts4 = new TreeSet<>((s, t) -> t.compareTo(s));
ts4.addAll(data);
```

HashMap<K, V>

- implements Map<K,V>
- Basata sull'ADT hash table, O(1), nessun ordine
- Mappa una chiave K (unica) ad un valore V
- Ctors:
 - HashMap() // vuota, capacity 16, load factor .75
 - HashMap(int) // capacity
 - HashMap(int, float) // capacity e load factor
 - HashMap(Map<? extends K, ? extends V>) // copy

TreeMap<K,V>

- implements NavigableMap<K,V>
- Basata sull'ADT albero → ordine, $O(\log(N))$
- Gli elementi inseriti devono implementare Comparable ed essere tutti mutualmente comparabili
- Ctors:
 - TreeMap() // vuota, ordine naturale
 - TreeMap(Comparator<? super K>) // sort by comparator
 - TreeMap(Map<? extends K, ? extends V>) // copy
 - TreeMap(SortedMap<K, ? extends V>) // copy + comparator

Reflection

- Package `java.lang.reflect`
- Permette di ottenere a run time informazioni su di una classe
- “Class” è la classe che rappresenta una classe
- “Field” rappresenta una proprietà, “Method” un metodo, ...

```
Class<?> c = Integer.class;  
Method[] methods = c.getMethods();  
for(Method method: methods) {  
    System.out.println(method);  
}
```

Tutti i metodi

Una specifica proprietà

```
Field field = ArrayList.class.getDeclaredField("elementData");  
field.setAccessible(true);  
Object[] data = (Object[]) field.get(al);
```

Multithreading

- Multitasking process-based vs thread-based
- L'interfaccia Runnable dichiara il metodo run()
- La classe Thread:
 - Ctors per Runnable
 - In alternativa, si può estendere Thread e ridefinire run()
 - start() per iniziare l'esecuzione

synchronized

- Metodo: serializza su this
- Blocco: serializza su oggetto specificato

comunicazione tra thread

- wait()
- notify() / notifyAll()