Java SE: Collezioni, …

- Inner class
- Generic
  - L’interfaccia Comparable
- Iteratori
- Collezioni
- Reflection
- Progetto di riferimento
  - https://github.com/egalli64/jse *(modulo 9)*

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
  - Se statica, solo i membri statici
- È possibile definirla come locale ad un blocco
- Inner class: non-static nested class
- Usate (ad es.)
  - Strutture dati complesse per la gestione dei dettagli implementativi interni
    - Liste, alberi: nodo
    - Mappe (dizionari): relazione chiave / valore
  - Swing (framework GUI): gestione degli eventi

Generic

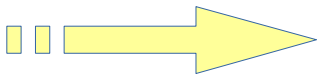
- Ci permette di avere classi che operano allo stesso modo su tipi differenti
  - Approccio alternativo all'ereditarietà
  - Più naturale se quello che cambia è il tipo solo di riferimento (es: collezioni)
- Migliora la sicurezza del codice rispetto al tipo di dato
  - Il tentativo di inserire oggetti di classe inattesa in una collezione sono intercettati in compilazione
- Il tipo utilizzato è indicato tra **parentesi angolari** (minore, maggiore)
  - ArrayList**<**String**>** è un ArrayList di stringhe
  - È possibile anche indicare più tipi, es: HashMap<Integer, String>
- Introdotti in Java 5 → limitazioni per compatibilità con versioni precedenti (type erasure)
  - Gestiti **solo reference**, i primitivi sono gestiti via autoboxing, usando il wrapper corrispondente
  - Non è possibile usare l'operatore instanceof, fare un cast, creare un array, …

L’interfaccia Comparable

- Se una classe implementa Comparable, i suoi oggetti sono ordinabili
  - È stata resa generica per il tipo con cui avviene il confronto
    - In questo modo, quando definiamo la classe, indichiamo con chi può essere confrontata
    - Tipicamente con la classe stessa
- Unico metodo: **compareTo**()
  - Di solito i valori di ritorno sono [-1, 0, 1], più in generale x.compareTo(y) ritorna
    - Un valore negativo → x < y
    - Zero → x == y
    - Un valore positivo → x > y
- Di solito, chi usa la classe si aspetta che valga anche
  - x.compareTo(y) == 0 → x.equals(y)
  - È buona norma assicurarlo, se non ci sono forti motivi per non farlo

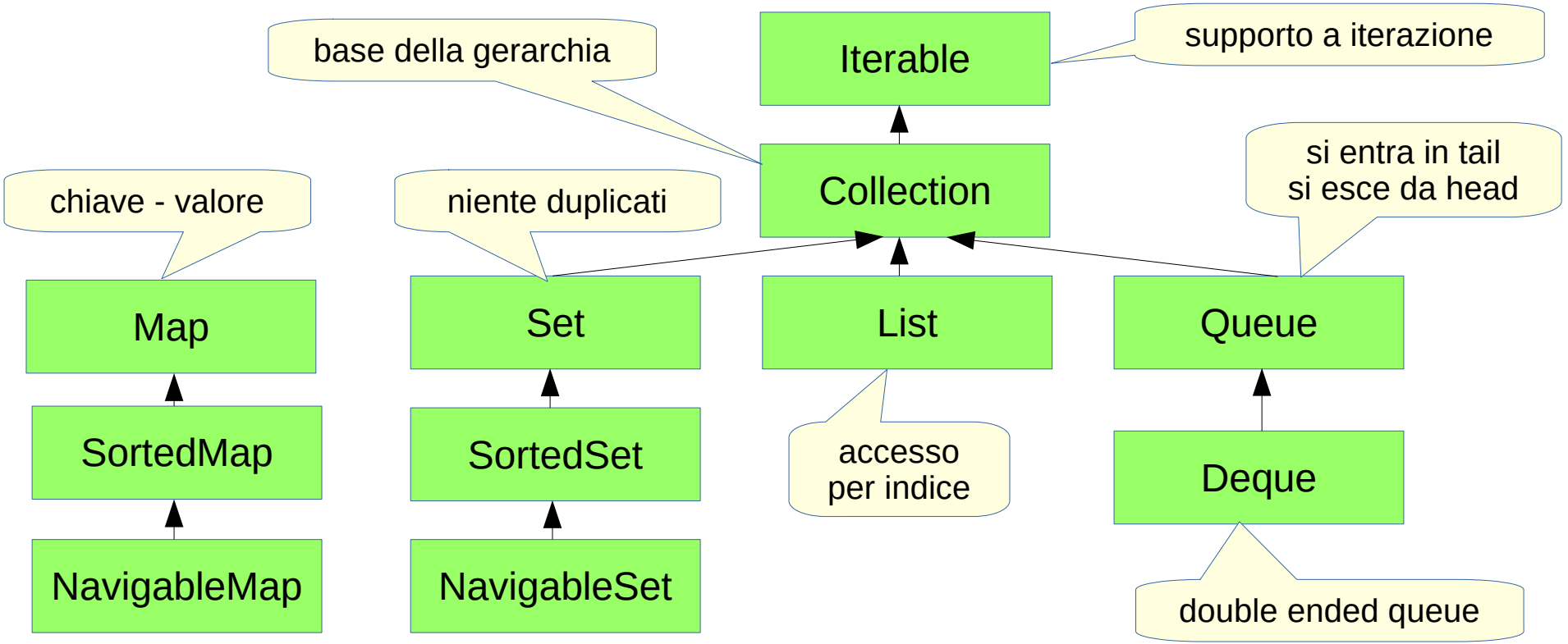
Java Collections Framework

- Gestione di gruppi di oggetti (solo reference!) seguendo principi di
  - Efficienza, performance, interoperabilità, estensibilità, adattabilità
- Basate su alcune interfacce standard



- Tutte le collezioni implementano Iterable, e dunque definiscono
  - **iterator**() che ritorna un reference a 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()

- Conversione di una collezione in un array
  - Più semplice: Object[] toArray()
  - Preferita: <T> T[] toArray(T[])
    - Il chiamante passa come argomento l'array di tipo e dimensione attesa che sarà ritornato

Alcuni metodi in List

- 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
- ListIterator<E> listIterator() // ritorna un iteratore con funzionalità specifiche
  - Permette di modificare l'elemento corrente, avere informazioni su elementi adiacenti
- List<E> of(), of(E), of(E, E), …, of(E …) // (Java 9)
  - Static factory method, ritorna una lista immutabile
- List<T> Arrays.asList(T …) // mutabile ma dimensione fissa (pre Java 9)

Alcuni metodi in Set e SortedSet

- In Set sono disponibili due static factory method
  - Set<E> of(), of(E), of(E, E), …, of(E …) // (Java 9)
  - Set<E> copyOf(Collection <? extends E>) // (Java 10)
  - Non accettano null, i risultati sono immutabili
- Per i SortedSet sono disponibili, tra gli altri, i seguenti metodi
  - E first()
  - E last()
  - SortedSet<E> subSet(E, E)
    - Intervallo chiuso a sinistra, aperto a destra

Alcuni metodi in NavigableSet

- 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

- In una coda, gli elementi si aggiungono a destra (tail) e si leggono/rimuovono a sinistra (head)
- 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

- 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

- 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

- Implementa l’interfaccia List, basandosi su un array
- Wrapper di un array standard, ma offre funzionalità da array dinamico
  - Ci pensa la classe a gestire lo spazio su disco
    - Possiamo specificare la dimensione iniziale dell’array (capacity) ma poi raramente ci interessa
      - Inserimento / eliminazione di elementi può determinare un cambiamento della capacità
    - Numero di elementi nella collezione è size – tutte le collezione hanno un metodo size()
- È la collezione più semplice, preferita in assenza di requisiti specifici
- Ctors
  - ArrayList() // capacity = 10
  - ArrayList(int) // set capacity
  - ArrayList(Collection<? extends E>) // copy

LinkedList

- Implementa le interfacce List e Deque (e dunque anche Queue)
- Lista doppiamente linkata
  - Head e tail permettono accesso diretto agli estremi
  - Memoria allocata solo per gli elementi utilizzati
    - E per reference al precedente e successivo di ogni elemento
- Andrebbe usata solo in casi d’uso particolari
  - Ad esempio, se si itera comunemente con inserimenti e eliminazioni (non solo agli estremi)
- Ctors
  - LinkedList() // vuota
  - LinkedList(Collection<? extends E>) // copy

ArrayDeque

- Implementa l’interfaccia Deque (e dunque anche Queue)
- Basato su un array circolare, dimensione dinamica
- Per la sua velocità è la collezione preferita per stack e code
- Ctors
  - ArrayDeque() // capacity = 16
  - ArrayDeque(int) // set capacity
  - ArrayDeque(Collection<? extends E>) // copy

HashSet

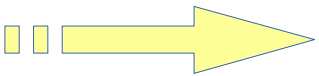
- Implementa l’interfaccia Set
- Basata su hash table
  - Non possiamo aspettarci che i suoi elementi siano in alcun ordine
  - Ma **O(1)** per add(), contains(), remove()
- Ctors:
  - HashSet() // vuota, capacity 16, load factor .75
  - HashSet(int) // capacity
  - HashSet(int, float) // capacity e load factor
  - HashSet(Collection<? extends E>) // copy

LinkedHashSet

- Deriva dalla classe HashSet
- 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

- Implementa l'interfaccia NavigableSet
- Basata sul tipo albero (BST) → ordine, O(log(N))
- Gli elementi inseriti devono implementare l'interfaccia 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

- Implementa l’interfaccia Map
- Le chiavi sono in un HashSet → O(1), nessun ordine
- Nessuna assunzione sulla collezione di valori
- 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

- Implementa l’interfaccia NavigableMap
- Le chiavi sono in un TreeSet
  - BST → ordine, O(log(N))
  - Elementi devono essere Comparable e tutti mutualmente comparabili
- Nessuna assunzione sulla collezione di valori
- 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);
```