

Programiranje 1

Poglavje 10: Vsebovalniki

Luka Fürst

Vsebovalnik

- podatkovna struktura, namenjena hranjenju elementov
- hierarhija vmesnikov in razredov v paketu `java.util`
- vsi vmesniki in razredi v hierarhiji so **generični**
 - tipni parameter = tip elementov v vsebovalniku
- tabela je tudi vsebovalnik, a ne pripada omenjeni hierarhiji

Osnovni tipi vsebovalnikov

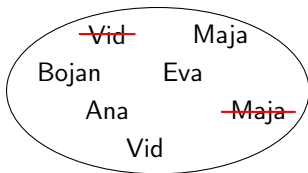
- seznam

- položaj elementa je določen z indeksom
- elementi se lahko podvajajo

Ana	Vid	Maja	Eva	Bojan	Maja	Vid
-----	-----	------	-----	-------	------	-----

- množica

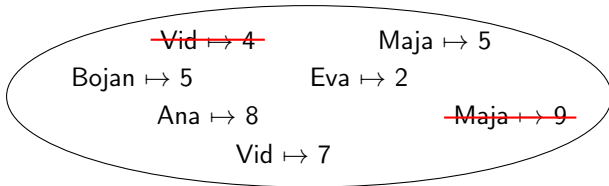
- položaj elementa ni določen (delna izjema: TreeSet)
- elementi se ne morejo podvajati



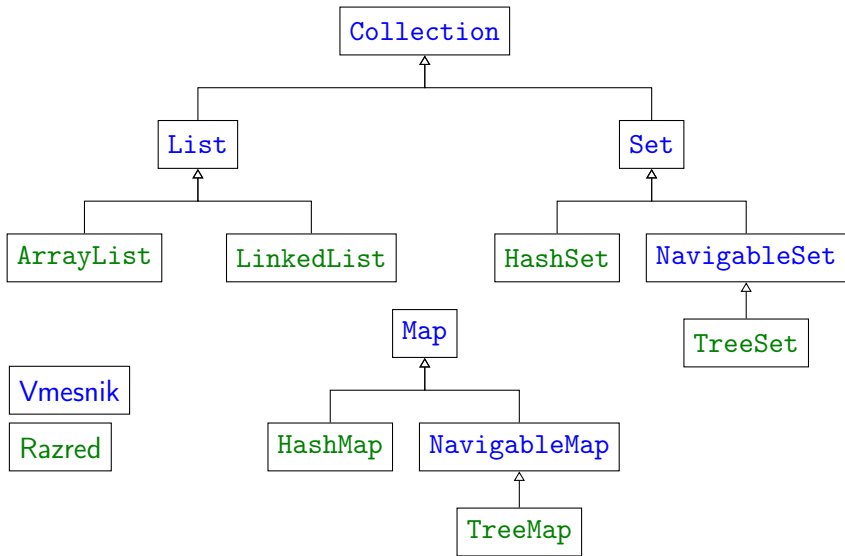
Osnovni tipi vsebovalnikov

- slovar

- množica parov ključ-vrednost
- ključi se ne morejo podvajati, vrednosti pa se lahko
- vrednosti so dostopne prek ključev



Hierarhija vsebovalnikov



Vmesnik Collection

- skupni vmesnik za **zbirke**
 - zbirka: seznam ali množica, ne pa tudi slovar
- v vmesniku `Collection` je deklarirana množica metod, predpisano pa je tudi njihovo privzeto obnašanje
- podvmesniki/podrazredi lahko obnašanje redefinirajo
 - `Collection`: add pomeni »dodaj element (kamorkoli)«
 - `List`: add pomeni »dodaj element na konec«

Izbor metod vmesnika Collection

- `boolean add(T element)`
Doda podani element v zbirko `this`.
- `boolean addAll(Collection<T> collection)`
Doda vse elemente iz zbirke `collection` v zbirko `this`.
- `boolean contains(T element)`
Vrne `true` natanko v primeru, če zbirka `this` vsebuje element `x` z lastnostjo `x.equals(element)`.
- `boolean containsAll(Collection<T> collection)`
Vrne `true` natanko v primeru, če v zbirki `this` za vsak element iz zbirke `collection` obstaja element `x` z lastnostjo `x.equals(element)`.

Izbor metod vmesnika Collection

- `boolean remove(T element)`

Iz zbirke `this` odstrani element `x` z lastnostjo `x.equals(element)`.

- `boolean removeAll(Collection<T> collection)`

Iz zbirke `this` odstrani vse elemente `x`, za katere v zbirki `collection` obstaja element `y` z lastnostjo `x.equals(y)`.

- `boolean retainAll(Collection<T> collection)`

V zbirki `this` ohrani samo tiste elemente `x`, za katere v zbirki `collection` obstaja element `y` z lastnostjo `x.equals(y)`.

- `void clear()`

Odstrani vse elemente iz zbirke `this`.

Izbor metod vmesnika Collection

- `boolean isEmpty()`
Vrne true natanko v primeru, če je zbirka `this` prazna.
- `int size()`
Vrne število elementov v zbirki `this`.
- `Iterator<T> iterator()`
Vrne iterator po elementih zbirke `this`.
- `String toString()`
Vrne vsebino zbirke v obliki niza $[e_1, \dots, e_n]$.
- `Object[] toArray()`
Izdela in vrne tabelo z elementi zbirke `this`.

Primer

```
Collection<Integer> zbirka = new HashSet<>();
zbirka.add(10);
zbirka.add(20);
zbirka.addAll(List.of(30, 40, 50));
System.out.println(zbirka);    // [50, 20, 40, 10, 30]

zbirka.remove(20);
System.out.println(zbirka.contains(20));    // false
System.out.println(zbirka.containsAll(Set.of(40, 10)));
                                                    // true

zbirka.retainAll(List.of(30, 50, 60));
System.out.println(zbirka);    // [50, 30]
System.out.println(zbirka.size());    // 2

for (Integer element: zbirka) {
    System.out.println(element);    // najprej 50, potem 30
}
```

Vmesnik List

- skupni vmesnik za sezname
- `interface List<T> extends Collection<T>`
- poleg metod vmesnika `Collection` ponuja tudi metode, ki sprejemajo indeks
- obnašanje nekaterih metod je redefinirano
 - npr. metoda `add` doda element na konec seznama, ne zgolj »nekam« v seznam

Izbor dodanih metod vmesnika List

- `void add(int index, T element)`
Vstavi podani element na podani indeks.
- `T get(int index)`
Vrne element na podanem indeksu.
- `int indexOf(Object obj)`
Vrne indeks prvega elementa `x`, za katerega velja `x.equals(obj)`, oziroma `-1`, če takega elementa ni.
- `T remove(int index)`
Odstrani element na podanem indeksu. Vrne odstranjeni element.

Izbor dodanih metod vmesnika List

- `void sort(Comparator<T> comp)`
Uredi seznam glede na primerjalnik `comp`. Če je `comp` enak `null`, uredi seznam glede na naravno urejenost (implementacijo vmesnika `Comparable<T>`).
- `static <T> List<T> of(T... elements)`
Ustvari **nespremenljiv** seznam s podanimi elementi.

Primer

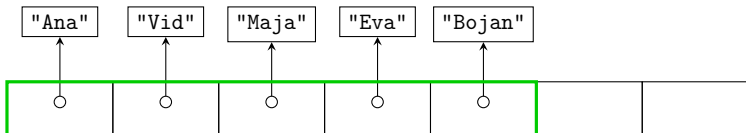
```
List<Cas> seznam = new LinkedList<>();
seznam.add(new Cas(10, 35));
seznam.add(new Cas(4, 55));
seznam.add(0, new Cas(15, 10));
seznam.add(1, new Cas(21, 20));
System.out.println(seznam); // [15:10, 21:20, 10:35, 4:55]

seznam.remove(1);
System.out.println(seznam); // [15:10, 10:35, 4:55]
System.out.println(seznam.size()); // 3

System.out.println(seznam.indexOf(new Cas(10, 20))); // -1
System.out.println(seznam.indexOf(new Cas(4, 55))); // 2
```

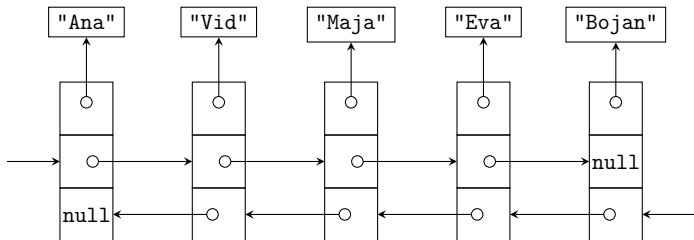
Razreda ArrayList in LinkedList

- implementaciji vmesnika List
- razred `ArrayList`
 - »raztegljiva« tabela (kot naš Vektor)
 - učinkovit dostop do elementa na podanem indeksu
 - dodajanje in odvzemanje elementov je učinkovito samo na koncu



Razreda ArrayList in LinkedList

- razred **LinkedList**
 - veriga elementov, dvosmerno povezana s kazalci
 - dostop do elementa na podanem indeksu ni učinkovit
 - dodajanje in odzemanje elementov je učinkovito kjerkoli



- **LinkedList** uporabimo, če pričakujemo veliko dodajanj in odzemanj na poljubnih položajih
- sicer uporabimo **ArrayList**

Vmesnik Set

- skupni vmesnik za množice
- `interface Set<T> extends Collection<T>`
- množica ne vsebuje podvojitev
- preverjanje podvojitev je odvisno od implementacije
 - pri razredu HashSet temelji na metodah equals in hashCode
 - pri razredu TreeSet temelji na implementaciji vmesnika Comparable ali Comparator
- vmesnik Set ponuja poleg metod vmesnika Collection tudi metodo

`static <T> Set<T> of(T... elements)`

ki ustvari nespremenljivo množico s podanimi elementi

Razred HashSet

- implementacija množice z zgoščeno tabelo
- razred, ki mu pripadajo elementi, mora implementirati metodi `hashCode` in `equals`
- veljati mora
$$x.equals(y) \implies (x.hashCode() == y.hashCode())$$
- čimvečkrat naj velja tudi obratno
- element `y` se obravnava kot dvojniki elementa `x`, če velja
$$x.equals(y) \text{ (in seveda tudi } x.hashCode() == y.hashCode())$$

Primer

- CasBrezEquals
 - različica razreda Cas brez metod equals in hashCode
- CasZEquals
 - različica razreda Cas z metodama equals in hashCode

```
Set<CasBrezEquals> mnozica1 = new HashSet<>();  
mnozica1.add(new CasBrezEquals(10, 35));  
mnozica1.add(new CasBrezEquals(10, 35));  
mnozica1.add(new CasBrezEquals(21, 20));  
System.out.println(mnozica1); // [21:20, 10:35, 10:35]
```

```
Set<CasZEquals> mnozica2 = new HashSet<>();  
mnozica2.add(new CasZEquals(10, 35));  
mnozica2.add(new CasZEquals(10, 35));  
mnozica2.add(new CasZEquals(21, 20));  
System.out.println(mnozica2); // [21:20, 10:35]
```

Razred TreeSet

- **urejena** množica, implementirana z dvojiškim iskalnim drevesom
- urejenost temelji na implementaciji vmesnika Comparable ali Comparator
- če uporabimo konstruktor `TreeSet()`, potem mora razred, ki mu pripadajo elementi, implementirati vmesnik Comparable
- če uporabimo konstruktor `TreeSet(Comparator<T> comp)`, potem je urejenost določena z metodo `compare` objekta `comp`
- podvojitve se prav tako preverjajo z metodo `compareTo` oz. `compare`
- $x.compareTo(y) == 0 \implies$ element `y` je podvojitev elementa `x`
- metoda `equals` se ne uporablja!

Razred TreeSet

- razred TreeSet implementira vmesnik NavigableSet, ki poleg metod vmesnika Set ponuja še nekatere metode, ki temeljijo na urejenosti
- `T first()`
Vrne najmanjši element množice.
- `T floor(T element)`
Vrne največji element množice, ki ni večji od podanega elementa, oziroma `null`, če takega elementa ni.

Razred TreeSet

- `NavigableSet<T> headSet(T element, boolean inclusive)`
Vrne podmnožico, ki vsebuje elemente, strogo manjše (`inclusive == false`) oziroma manjše ali enake (`inclusive == true`) od podanega elementa. Kasnejše spremembe množice `this` vplivajo na vrnjeno množico (in obratno).
- `T last()`
`T ceiling(T element)`
`NavigableSet<T> tailSet(T element, boolean inclusive)`
Delujejo enako kot `first`, `floor` in `headSet`, le da namesto (naj)manjših elementov izbirajo (naj)večje in obratno.

Primer

```
// razred Cas implementira Comparable<Cas>

NavigableSet<Cas> mnozica = new TreeSet<>();
mnozica.addAll(List.of(
    new Cas(10, 35), new Cas(10, 35), new Cas(6, 50),
    new Cas(5, 15), new Cas(23, 40), new Cas(12, 0)));

System.out.println(mnozica); // [5:15, 6:50, 10:35, 12:00, 23:40]
System.out.println(mnozica.last()); // 23:40
System.out.println(mnozica.floor(new Cas(13, 0))); // 12:00
Set<Cas> pod = mnozica.headSet(new Cas(12, 0), false);
System.out.println(pod); // [5:15, 6:50, 10:35]

pod.remove(new Cas(6, 50));
System.out.println(pod); // [5:15, 10:35]
System.out.println(mnozica); // [5:15, 10:35, 12:00, 23:40]
```

Vmesnik Map

- skupni vmesnik za slovarje
- `interface Map<K, V>`
- K: tip ključev
- V: tip vrednosti
- ključi se ne podvajajo, vrednosti pa se lahko
- `HashMap`
 - unikatnost ključev glede na `equals`
 - za noben par ključev ne velja `x.equals(y)`
- `TreeMap`
 - unikatnost ključev glede na primerjalnik
 - za noben par ključev ne velja `x.compareTo(y) == 0` oziroma `primerjalnik.compare(x, y) == 0`

Izbor metod vmesnika Map

- `V put(K key, V value)`

V slovar doda podani par ključ-vrednost oziroma zamenja vrednost, ki pripada ključu `key`, če ta v slovarju že obstaja.

- `V get(Object key)`

Vrne vrednost, ki pripada podanemu ključu, oziroma `null`, če slovar ne vsebuje podanega ključa.

- `boolean containsKey(Object key)`

Vrne `true` natanko v primeru, če slovar vsebuje podani ključ.

- `V remove(Object key)`

Poišče ključ `x` z lastnostjo `x.equals(key)` ter odstrani ključ `x` in pripadajočo vrednost.

Izbor metod vmesnika Map

- `boolean isEmpty()`
Vrne true natanko v primeru, če je slovar prazen.
- `void clear()`
Izprazni slovar.
- `int size()`
Vrne število parov ključ-vrednost (= število ključev).
- `Set<K> keySet()`
Vrne množico ključev.
- `Set<Map.Entry<K, V>> entrySet()`
Vrne množico parov ključ-vrednost.
- `Collection<V> values()`
Vrne zbirko vrednosti.

Razreda HashMap in TreeMap

- enakovredna razredoma HashSet in TreeSet, če slovar obravnavamo kot množico parov ključ-vrednost
- **HashMap**
 - neurejen slovar
 - unikatnost ključev je določena z metodo equals
- **TreeMap**
 - slovar, urejen po ključih
 - urejenost in unikatnost ključev sta določeni z implementacijo vmesnika Comparable ali Comparator
 - TreeMap implementira vmesnik NavigableMap, ki ponuja še nekatere metode, temelječe na urejenosti ključev

Primer

```
// v razredu Cas sta redefinirani metodi equals in hashCode
Map<Cas, String> opravki = new HashMap<>();
opravki.put(new Cas(8, 15), "predavanja");
opravki.put(new Cas(11, 10), "govorilne ure");
opravki.put(new Cas(12, 30), "kosilo");
opravki.put(new Cas(14, 0), "sestane");
opravki.put(new Cas(18, 30), "večerja");
System.out.println(opravki); // {18:30=večerja, 14:00=sestane, ...}

System.out.println(opravki.get(new Cas(12, 0))); // null
System.out.println(opravki.get(new Cas(14, 0))); // sestane
System.out.println(opravki.containsKey(new Cas(8, 15))); // true
opravki.remove(new Cas(14, 0));
System.out.println(opravki);
// {18:30=večerja, 11:10=govorilne ure, 12:30=kosilo, 8:15=predavanja}

for (Cas cas: opravki.keySet()) {
    System.out.printf("%s -> %s%n", cas, opravki.get(cas));
}
// 18:30 -> večerja
// 11:10 -> govorilne ure
// ...
```

Še en primer

```
// razred Cas implementira vmesnik Comparable<Cas>
Map<Cas, String> opravki = new TreeMap<>(Map.of(
    new Cas(8, 15), "predavanja",
    new Cas(11, 10), "govorilne ure",
    new Cas(12, 30), "kosilo",
    new Cas(14, 0), "sestane",
    new Cas(18, 30), "večerja"
));

for (Cas cas: opravki.keySet()) {
    System.out.printf("%s -> %s\n", cas, opravki.get(cas));
}

// 8:15 -> predavanja
// 11:10 -> govorilne ure
// 12:30 -> kosilo
// 14:00 -> sestane
// 18:30 -> večerja

for (Map.Entry<Cas, String> par: opravki.entrySet()) {
    System.out.printf("%s -> %s\n", par.getKey(), par.getValue());
}

// še en tak izpis
```

Razred Collections

- vsebuje razne statične metode za delo z vsebovalniki
- `static <T> int binarySearch(List<T> list, T key, Comparator<T> comp)`

S pomočjo dvojiškega iskanja poišče element `key` v seznamu `list`, ki mora biti urejen glede na primerjalnik `comp`. Če element obstaja, vrne njegov indeks, sicer pa vrne $(-p - 1)$, kjer je p indeks, kamor bi iskani element sodil.

- `static <T> T max(Collection<T> coll, Comparator<T> comp)`

Vrne največji element podane zbirke glede na podani primerjalnik.

- `static <T> void shuffle(List<T> list)`

Naključno premeša elemente podanega seznama.