Programozási nyelvek Java

12.gyakorlat

Gyűjtemény keretrendszer (Collections Framework)

A java.util.* csomag részei.
Objektumok memóriában való tárolására, lekérdezésére és manipulálására szolgálnak.

Általános célú adatszerkezetek

- Collection, List, Deque
- Set, SortedSet
- Map, SortedMap

Nem megvalósított művelet UnsupportedOperationException-t dob.
Copy konstruktorok vannak (egyik a másikra konvertálható).

Műveletek csoportjai

1. Alapvető műveletek: size(), isEmpty(), contains(), add(), remove()
2. Elemek együttes kezelése: addAll(), containsAll(), removeAll(), clear(), retainAll()
3. Tömbbé konvertálás:
   - A[] arr = (A[]) list.toArray(new A[list.size()]);
   - A[] arr = (A[]) list.toArray();

Iterátorokkal rendelkeznek, használhatók a collection-önös for ciklusban.

Halmaz

Duplikált elemeket nem tartalmazhat.
Szükséges hozzá az objektumon az equals() (2 halmaz egyenlő, ha ugyanazokat az elemeket tartalmazzák) és a hashCode() (hash-elő implementációk, nem számít a sorrend) metódusok felüldefiniálása.

HashSet

- hatékonyabb, mint a TreeSet
- **void add(Object o)**: hozzáad egy elemet a halmazhoz
- **void remove(Object o)**: töröl egy elemet a halmazból
- **void clear()**: tölri a halmaz tartalmát
- **int size()**: megadja a halmaz méretét
- **boolean isEmpty()**: visszaadja, hogy üres-e a tömb
- **boolean contains(Object o)**: megivzsgálja, hogy tartalmazza-e az elemet

TreeSet

- kevésbé hatékony, de rendezett
- **void add(Object element)**: hozzáad egy elemet
- **void addAll(Collection elements)**: hozzáadja az egész tartalmat
- **void remove(Object element)**: töröl egy elemet
- **void clear()**: törli a tartalmat
- **int size()**: megadja a méretét
- **boolean isEmpty()**: visszaadja, hogy üres-e
- **boolean contains(Object element)**: megvizsgálja, hogy tartalmazza-e az elemet
- **Object first()**: visszatér a legkisebb elemmel
- **Object last()**: visszatér a legnagyobb elemmel
- **SortedSet headSet(Object toElement)**: a megadott elemig visszatér az összessel
- **SortedSet tailSet(Object fromElement)**: a megadott elemtől visszatér az összessel
- **SortedSet subSet(Object fromElement, Object toElement)**: visszatér a köztes elemekkel

Lista

Elemek tárolására, pozíció szerinti elérésére, iterációjára és részlista kezelésére szolgálnak.

LinkedList<T>

- **void add(Object element)**: hozzáad egy elemet
- **void add(int index, Object element)**: beszúr egy elemet az adott helyre
- **void addFirst(Object element)**: beszúrja a lista legelejére a megadott elemet
- **void addLast(Object element)**: beszúrja a lista legvégére a megadott elemet
- **void addAll(Collection elements)**: hozzáadja az egész tartalmat
- **void addAll(int index, Collection elements)**: beszúrja az elemeket a megadott indextől kezdve
- **void remove(int index)**: törli a megadott indexű elemet
- **void remove(Object element)**: törli az adott elemet
- **void removeFirst()**: törli az első elemet
- **void removeLast()**: törli az utolsó elemet
- **void clear()**: törli a tartalmat
- **int size()**: megadja a méretét
- **boolean isEmpty()**: visszaadja, hogy üres-e
- **boolean contains(Object element)**: megvizsgálja, hogy tartalmazza-e az elemet
- **Object get(int index)**: visszatér az adott pozícióban levő elemmel
- **Object getFirst()**: visszatér a legelső elemmel
- **Object getLast()**: visszatér az utolsó elemmel
- **int indexOf(Object element)**: visszatér a megadott elem indexével
- **int lastIndexOf(Object element)**: visszatér az utolsó elemmel, ami egyenlő vele
- **void set(int index, Object element)**: átállítja az adott indexű elemet a megadottra

Map

- Kulcs-érték párokat tartalmazó adatszerkezet
- Minden kulcshoz egy érték tartozhat
- Nem iterálható, azonban lekérdezhető a keySet() és az values(), ami már igen
- HashMap, Hashtable
- ConcurrentMap, SortedMap

Hashtable

Szükséges hozzá az objektumon az equals() metódusok felüldefiniálása.
Hiszen a kulcsot elhasj-eli, és az alapján tárolja el

- Konstruktora lehet üres, de megadható maximális kapacitása is.
- **V put(K key, V value)**: beszúrja a megadott kulccsal a megadott értéket
- **void putAll(Map<? extends K, ? extends V> m)**: beszúr egy map típusú adatszerkezetet
- **V remove(Object key)**: törli a megadott elemet
- **V get(Objetc key)**: visszatér az adott kulcsú elemmel, ha nincs ilyen, akkor null-lal
- **Set<K> keySet()**: visszatér a kulcsok halmazával
- **Enumeration<K> keys()**: visszatér a kulcsok enumerációjával
- **Collection<V> values()**: visszatér az értékekkel
- **Enumeration<V> elements()**:visszatér az értékek enumerációjával
- **boolean contains(Object element)**: szerepel-e a megadott elem az értékek között
- **boolean containsKey(Object key)**: szerepel-e a megadott kulcs a kulcsok között
- **boolean containsValue(Object element)**: szerepel-e a megadott elem az értékek között
- **void clear()**: kitörli a tartalmát
- **boolean isEmpty()**: megmondja, hogy üres-e
- **int size()**: megadja az elemek számát

HashMap

Szükséges hozzá az adott típusú objektumon az equals() metódusok felüldefiniálása.

- Konstruktora lehet üres, de megadható maximális kapacitása is.
- **V put(K key, V value)**: beszúrja a megadott kulccsal a megadott értéket
- **void putAll(Map<? extends K, ? extends V> m)**: beszúr egy map típusú adatszerkezetet
- **V remove(Object key)**: törli a megadott elemet
- **V get(Objetc key)**: visszatér az adott kulcsú elemmel, ha nincs ilyen, akkor null-lal
- **Set<K> keySet()**: visszatér a kulcsok halmazával
- **Collection<V> values()**: visszatér az értékekkel
- **boolean containsKey(Object key)**: szerepel-e a megadott kulcs a kulcsok között
- **boolean containsValue(Object element)**: szerepel-e a megadott elem az értékek között
- **void clear()**: kitörli a tartalmát
- **boolean isEmpty()**: megmondja, hogy üres-e
- **int size()**: megadja az elemek számát

Kényelmi lehetőségek

- **java.util.Arrays**
  - **asList(Object[] t)**: tömbből listát készít
  - **sort(Object[] t)**: összefésüléses rendezés (feljavított), sok túlterhelés
- **java.util.Collections**
  - **Set singleton(Object o)**: egyelemű, módosíthatatlan halmaz
  - **List singletonList(Object o)**: egyelemű, módosíthatatlan lista
  - **Map singletonMap(Object o)**: egyelemű, módosíthatatlan map
  - **copy(List dest, List src)**: másolás
  - **Object min(Collection coll)**: megadja a minimális elemet
  - **Object min(Collection coll, Comparator c)**: megadja a minimális elemet
  - **Object max(Collection coll)**: megadja a maximális elemet
  - **Object max(Collection coll, Comparator c)**: megadja a maximális elemet
  - **void reverse(List l)**: megfordítja a listát
  - **void shuffle(List l)**: megkeveri a lista elemeit
  - **void swap(List l, int i, int j)**: megcseréli a két elemet
  - **void sort(List l)**: rendezi a listát (az Arrays.sort függvényt hívja meg + konvertál)

Megjegyzések

- Az interfész műveleteken kívül rengeteg egyéb hasznos funkcionalitással rendelkezik, érdemes a javadocot olvasgatni
- Saját implementációk!!!☺ A Collections Framework absztrakt osztályokat biztosít (AbstractList, AbstractSet, …), amikből lehet származtatni.
- További adatszerkezetek: Dequeue, Stack, BitSet, …
- Célszerű minél általánosabb interfészt megadni
  List<Integer> v2 = new LinkedList<Integer>(); // listakent kezelés
- Részletesen: http://java.sun.com/javase/6/docs/api/java/util/package-summary.html

Feladatok

1. Készítsetek egy generikus függvényt, ami a paramétereiben kapott 2 halmaz metszetét adja vissza.
   Valósítsátok meg hasonlóan az unió műveletet is.

2. Készíts egy programot, amely megszámolja egy fájlban az egyes szavak előfordulásainak számát! A program a fájl elérési útját argumentumként kapja meg. A megvalósításhoz használj egy String ! Integer leképezést (Hashtable<String, Integer>)!

3. Készítsetek egy Date osztályt, amely tartalmazza az év, hónap, nap adatokat (mind számok). Implementáljátok vele a Comparable<Date> interfészt, és ennek megfelelően valósítsátok meg a compareTo() függvényt!
   Teszteléshez hozzatok létre kódból 3 Date típusú objektumot és tároljátok el egy tetszőleges lista adatszerkezetben. Majd rendezzétek kronológiai sorrend szerint és írjátok ki az eredményt a képernyőre.

4. Készíts menüt az előző feladathoz!
   Menü:
   - Új dátum felvétele
   - Dátumlista betöltése
   - Dátumlista mentése (ezt tudjuk be tölteni is)
   - Dátumok listázása időrendben a képernyőre
   - Kilépés

5. Készítsünk egy sorozat rendező alkalmazást!

   Készítsünk egy Series osztályt, ami tartalmazza a sorozat címét, az évadok számát, a részek számát, továbbá Episode típusú objektumokat, amik egy-egy részt reprezentálnak (a fájl 1 sora).
   Implementáljuk az Episode osztályban a Comparable<Episode> interfészt!
   A compareTo() működjön úgy, hogy elsődleges szempont szerint az évad, azon belül pedig az epizódszám alapján rendezzen!
   Mindkét osztályban implementáljuk az equals, toString és a hashCode függvényeket is!

   A tesztelő osztály egy menüvel induljon.
   Menü:
   - Új sorozat (elég csak a név megadása)
   - Új epizód felvétele (évad, epizódszám, cím)
   - Sorozat megjelenítése (epizódok rendezetten)
   - Sorozat betöltése
   - Sorozat mentése (ezt tudjuk betölteni is)
   - Kilépés

   Megjelenítéskor lehessen kiválasztani egy sorozatot és annak a részeit listázza ki időrendben (évaddal, epizódszámmal és címmel).
   Epizód felvételénél pedig egy kiválasztott sorozathoz lehessen felvenni új részt, évad, epizódszám és cím megadásával.

Mentéskor mindent mentsünk le (figyeljünk arra, ha több sorozatot kell elmenteni). A fájl a következő formátumú legyen (egy sorozat esetén):

South park
12:7:Super Fun Time
12:4:Canada on Strike
8:13:Cartman's Incredible Gift
10:8:Make Love, Not Warcraft

Első sor a sorozat címe, aztán a többi sor szerkezete: évad, epizódszám, epizódcím.