Java és web programozás

Kovács Kristóf, Rimay Zoé

Budapesti Műszaki Egyetem

2013. október 2.

4. Előadás

Adatszerkezetek

A java által definiált adatszerkezetek:

- Enumeration
- BitSet
- Vector
- Stack
- Dictionary
- Hashtable
- Properties

Adatszerkezetek

A java által definiált adatszerkezetek:

- Enumeration
- BitSet
- Vector
- Stack
- Dictionary
- Hashtable
- Properties

Az általunk használt java verzióban ezek már elavult, csak a korábbi verzióval való kód kompatibilitás miatt léteznek. De a jelenlegi adatszerkezetek lényegét ezeken keresztül lehet legjobban átadni.

Enumeration

- Az Enumeration valójában nem egy adatszerkezet, csak egy *interface* (késõbb).
- Gondolhatunk most erre úgy, mint osztályoknak egy tulajdonságára.
- Azok az osztályok amik *implementálják* ezt az *interface*-t képesek az általuk tárolt elemeket egymás után lekérdezni.

BitSet

- A BitSetre tekinthetünk úgy, mint egy *boolean* tömbre, ami dinamikusan nõ és csökken, tehát nem kell a méretével törõdnünk.
- Amikor felveszek egy új elemet a BitSetbe, annak új helyet foglal le. Amikor törlünk akkor sorban felszabadítja az üres helyeket.
- Hasznos, ha valamilyen bináris tulajdonságot szeretnénk tárolni.
- Egyszerû példa, ha egy igaz-hamis tesztet teszünk a honlapunkra, ami több oldalból áll. Az oldalak közt tarolnunk kell, hogy a korábbi oldalakon szereplõ kérdesekre mit valaszolt a felhasználó.

Vector

- A Vector nagyon hasonlít a szokásos tömbre, azzal a kivétellel, hogy dinamikusan változik a mérete, ha új elemeket helyezünk bele, vagy törlünk.
- Mint egy tömbben is, a Vector elemeit indexen keresztül is elérhetjük.
- A hasznosságáról azthiszem nem is kell többet mondani, könnyedén lecseréli majdnem az összes tömböt amit eddig használtunk.
- Az általunk leggyakrabban használt adatszerkezet lesz.

Stack

- A Stack egy LIFO (last in first out) megvalósitás.
- Úgy kell elképzelni, mint a valódi életben is amikor tárgyakat egymásra pakolunk. Mindig a legutolsót amit hozzáadtunk a Stackhez azt tudjuk csak levenni.
- Ritka esetekben fogjuk használni, vagy az is lehet hogy soha. Viszont fontos, hogy tudjatok a létezéséről.

Dictionary / Hashtable

- Ezt a kettõt egyben tárgyalnám.
- A Dictionary, mint ahogy az Enumeration is, csak egy *interface* így belõle nem lehet objektumot létrehozni.
- Viszont meghatározza, hogy milyen tulajdonságokkal kell rendelkeznie egy olyan osztálynak amit Dictionary-ként lehet használni.
- Azon adatszerkezetek a Dictionary-k, amik kulcsokhoz rendelnek értékeket. Tehát egy tömbbel ellentétben egy adott elemet nem az indexével, hanem a kulcsával tudunk elérni.
- Például, ha szeretnénk valamilyen gyors módot ahogy le lehet kérdezni userek teljes nevét, akkor vehetnénk a userneveket kulcsként és a teljes nevet értékként.
- A Hashtable egy konkrét implementációja ennek.

Properties

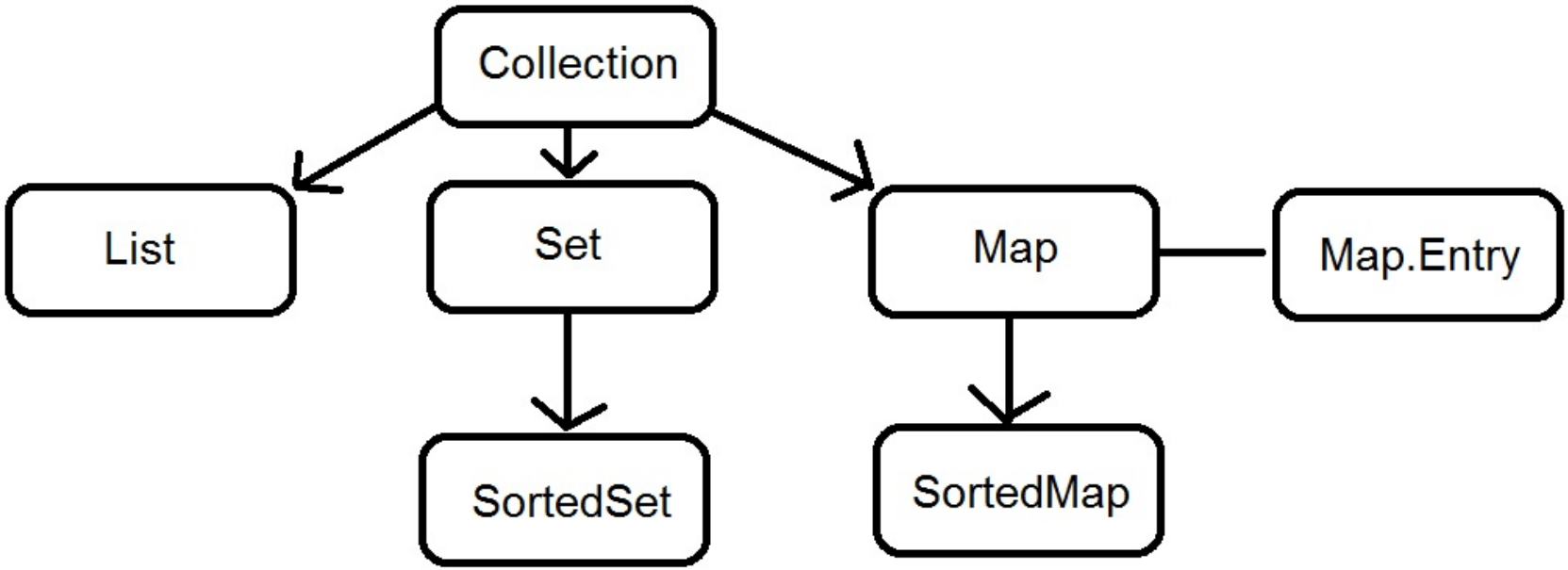
- A Properties egy leszármazottja a Hashtable-nek.
- Annyi speciális van benne, hogy mindig Stringekhez Stringeket rendel hozzá.
- Azért érdemes külön tárgyalni, mert sok beépített java lekérdezés ilyen objektumot ad vissza.
- Pl:

```
System.getProperties()
```

Felejtés

- Az elõzõ pár diából amit meg kell jegyezni az az ötlet ami mögöttük van, és nem a konkrét elnevezések.
- A java 2.0 óta ezeket teljesen átírták.
- Viszont az új megvalósításokat sokkal nehezebb lett volna bevezetni mélyebb java tudás nélkül.

Interface-ek



- Ezek mind csak *interface*-ek, tulajdonságok. Nem hozhatunk létre konkrét Set objektumot.
- Viszont lesznek majd olyan osztályok, amik rendelkeznek valamennyi tulajdonsággal ezek közül, belõlük már létrehozhatunk objektumokat.
- Amint látható hierarchiába vannak szervezve, mind Collection, és pl a SortedSet egy Set.

Collection

- Annyit tud, hogy objektumoknak valamilyen csoportját tárolja.

List

- A List rendezetten tárol elemeket.

Set

- A Set egyedi elemeket tárol.

SortedSet

- Rendezett és egyedi elemeket tárol.

Map

- Egyedi kulcsokhoz értékeket rendel

SortedMap

- A kulcsokat rendezetten tárolja.

Map.Entry

- Leír egy kulcs-értek párt. Ez a Mapnek egy belsõ osztálya.

Az osztályok

- Nem mindet fogom felsorolni, a többinek utána lehet nézni (van sok).
- Amiket felsorolok az általam gondolt leghasznosabbak, a félév során ezeket fogjuk használni:
  - ArrayList
  - HashSet
  - HashMap

Közös metódusok

Ezeket a metódusokat a Collection *interface* implementálja, így az elõbb említett összes adatszerkezetben mûködnek. Nem sorolok fel minden létezõ metódust, a továbbiakról eclipse-ben is olvashattok.

- boolean add(Object o)
  - A Collection végére beteszi az adott objektumot.
- boolean addAll(Collection c)
  - Az adott Collection összes elemét beteszi a Collectionbe.
- void clear()
  - Törli a Collection tartalmát.
- boolean contains(Object obj)
  - Igazat ad, ha az adott elemet tartalmazza a Collection, hamisat különben.
- boolean isEmpty()
  - Üres-e a Collection.

- Iterator iterator()
  - Visszaadja a Collection iterátorát (errõl késõbb).
- boolean remove(Object obj)
  - Töröl egy olyan elemet a Collectionbõl ami megegyezik a kapott *obj* objektummal. Hamisat ad, ha nem volt mit kitörölni.
- int size()
  - Visszaadja, hogy hány elem található a Collectionben.
- Object[] toArray()
  - Visszaadja a tárolt elemekbõl készitett tömböt.

ArrayList

- Ahogy a nevébõl ki lehet találni, ez egy List.
- Szinte az összes tömbünket le fogjuk cserélni erre.
- Dinamikusan nõ a mérete, így nem kell foglalkozunk a létrehozásakor azzal, hogy majd hány elem lesz benne.

Fontos metódusai:

- void add(int index, Object element)
  - Az adott indexre berakja az objektumot a már bentlevõket jobbra eltolja. (Nem irja felül az indexen találhato objektumot.)
- void ensureCapacity(int minCapacity))
  - Elõre lefoglal legalább ennyi helyet. Ha ezen túlnõ, tovább növeli a tárolt helyet.
- Object get(int index)
  - Visszaadja az adott indexen tárolt elemet.
- int indexOf(Object o)
  - Hanyadik indexen található az adott elem elõször.
- Object remove(int index)
  - Adott indexû elemet töröl.

- void removeRange(int fromIndex, int toIndex)
  - Két index között mindent töröl.
- Object set(int index, Object element)
  - Lecseréli az adott indexen találhato elemet egy új elemre.

```
import java.util.*;

public class ArrayListDemo {
   public static void main(String args[]) {
      // create an array list
      ArrayList al = new ArrayList();
      System.out.println("Initial size: " + al.size());
      // add elements to the array list
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");
      al.add(1, "A2");
```

```
    System.out.println("Size after additions: " + al.size());
    // display the array list
    System.out.println("Contents: " + al);
    // Remove elements from the array list
    al.remove("F");
    al.remove(2);
    System.out.println("Size after deletions: " + al.size());
    System.out.println("Contents: " + al);
  }
}
```

Eredmény:

```
Initial size: 0
Size after additions: 7
Contents: [C, A2, A, E, B, D, F]
Size after deletions: 5
Contents : [C, A2, E, B, D]
```

HashSet

- Egy Set, ami hash táblát használ az elemek tárolásához.
- Azok a metódusai, mint a Collectionnek.

```
import java.util.*;
public class HashSetDemo {
   public static void main(String args[]) {
      HashSet hs = new HashSet();
      // add elements to the hash set
      hs.add("B");
      hs.add("A");
      hs.add("D");
      hs.add("E");
      hs.add("C");
      hs.add("F");
      System.out.println(hs);
   }
}
```

Eredmény: [D, E, F, A, B, C]

HashMap

- Hasonlóan mûködik, mint pythonban a dictionary.
- Szintén hashelés segítségével tárolja az elemeket.

Fontos metódusai:

- boolean containsKey(Object key)
  - Tartalmazza-e az adott kulcsot.
- boolean containsValue(Object value)
  - Tartalmazza-e az adott érteket.
- Object get(Object key)
  - Az adott kulcshoz tartozó értéket lekéri.
- Object put(Object key, Object value)
  - Berakja ezt a kulcs-érték párt.
- Object remove(Object key)
  - Az adott kulcsú elemet törli.

```
import java.util.*;

public class HashMapDemo {
   public static void main(String args[]) {
      HashMap hm = new HashMap();
      // Put elements to the map
      hm.put("Zara", new Double(3434.34));
      hm.put("Mahnaz", new Double(123.22));
      hm.put("Ayan", new Double(1378.00));
      hm.put("Daisy", new Double(99.22));
      hm.put("Qadir", new Double(-19.08));

      // Get a set of the entries
      Set set = hm.entrySet();
      // Get an iterator
      Iterator i = set.iterator();
      // Display elements
}
```

```
        while(i.hasNext()) {
           Map.Entry me = (Map.Entry)i.next();
           System.out.print(me.getKey() + ": ");
           System.out.println(me.getValue());
        }
        // Deposit 1000 into Zara's account
        double balance = ((Double)hm.get("Zara")).doubleValue
        hm.put("Zara", new Double(balance + 1000));
        System.out.println("Zara's new balance: " +
        hm.get("Zara"));
    }
```

Eredmény:

```
Zara: 3434.34
Mahnaz: 123.22
Daisy: 99.22
Ayan: 1378.0
Qadir: -19.08
Zara's new balance: 4434.34
```

Iterátor

- Elég gyakori, hogy végig szeretnénk futni egy Collection elemein. Például egyenként ki akarjuk õket írni, vagy valami mûveletet végrehajtani rajtuk.
- Ekkor jönnek szóba az *iterátorok*
- Elõször el kell kérni az adott objektumtól az iterátorát, a már említett metódussal.

Iterátor

- Elég gyakori, hogy végig szeretnénk futni egy Collection elemein. Például egyenként ki akarjuk õket írni, vagy valami mûveletet végrehajtani rajtuk.
- Ekkor jönnek szóba az *iterátorok*
- Elõször el kell kérni az adott objektumtól az iterátorát, a már említett metódussal.
- Ezek az általános lépések az iterálásra:
  - Elkérjük az iterátort
  - Ciklust készitünk, melynek a feltétele, az iterátor *hasNext()* metódusa.
  - A cikluson belül az iterátor *next()* metódusával lekérjük a következõ elemet.
- Az iterátoroknak van *remove()* metódusa is, amivel az adott elemet törölhetjük a Collectionbõl.

```
import java.util.*;
public class IteratorDemo {
   public static void main(String args[]) {
      ArrayList al = new ArrayList();
      al.add("C");
      al.add("A");
      al.add("E");
      al.add("B");
      al.add("D");
      al.add("F");
      // Use iterator to display contents of al
      System.out.print("Original contents of al: ");
      Iterator itr = al.iterator();
      while(itr.hasNext()) {
         Object element = itr.next();
         System.out.print(element + " ");
      }
      System.out.println();
```

```
        // Modify objects being iterated
        ListIterator litr = al.listIterator();
        while(litr.hasNext()) {
            Object element = litr.next();
            litr.set(element + "+");
        }
        System.out.print("Modified contents of al: ");
        itr = al.iterator();
        while(itr.hasNext()) {
            Object element = itr.next();
            System.out.print(element + " ");
        }
        System.out.println();
    }
}
```

Eredmény:

```
Original contents of al: C A E B D F
Modified contents of al: C+ A+ E+ B+ D+ F+
```

Zárszó

- Az eddigi kódokban a Collectionjeinknek nem mondtuk meg, hogy mit tarolnak. Ez általában nem jó ötlet.
- A következõ módon lehet tudatni bármelyik Collectionnel, hogy egy adott dolgot tárol:

```
ArrayList<String> myList = new ArrayList<String>();
```

Zárszó

- Az eddigi kódokban a Collectionjeinknek nem mondtuk meg, hogy mit tarolnak. Ez általában nem jó ötlet.
- A következõ módon lehet tudatni bármelyik Collectionnel, hogy egy adott dolgot tárol:

```
ArrayList<String> myList = new ArrayList<String>();
```

- A String helyére persze bármilyen tipust írhatunk, akár általunk írt osztályt is.
- Sõt, még ezek is megtehetõk, errõl majd késõbb lesz magyarázat:

```
List<String> myList2 = new ArrayList<String>();

Collection<String> myList3 = new ArrayList<String>();
```