Java és web programozás

Kovács Kristóf

Budapesti Műszaki Egyetem

2015. 03. 18.

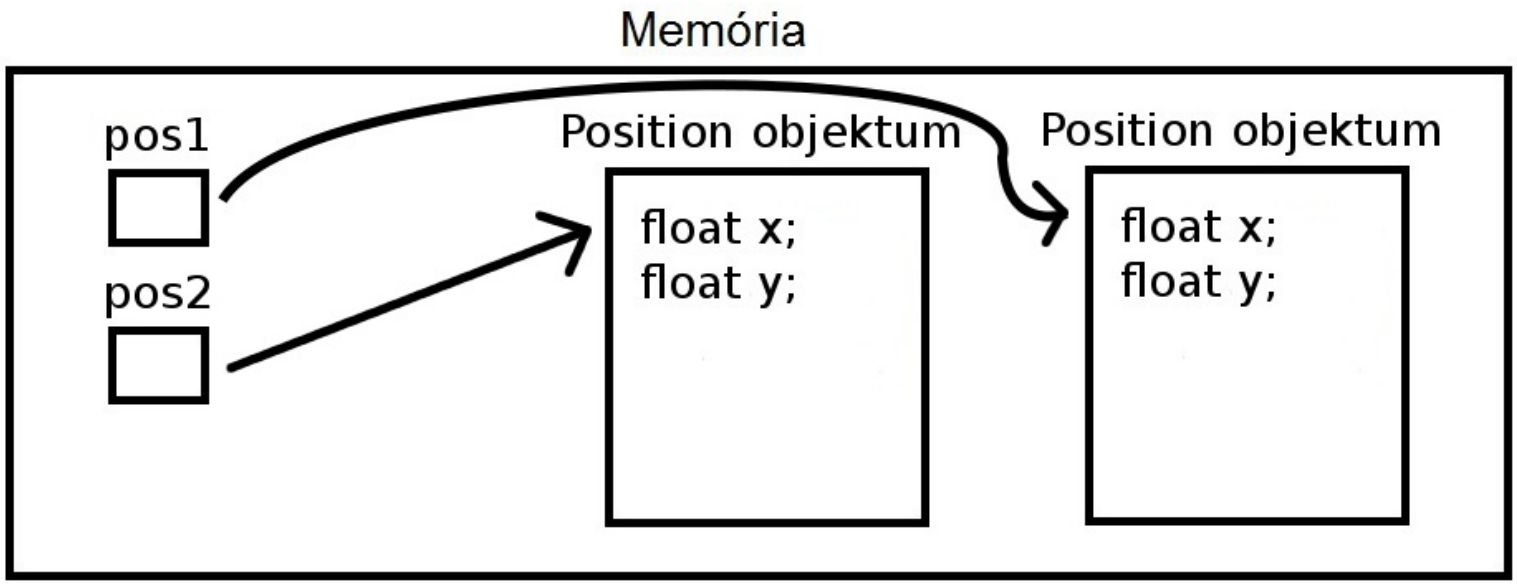
5. Előadás

Ismétlés: osztály, konstruktor, objektum

```
public class Complex {
    private float rePart_;
    private float imPart_;

    public Complex(float rePart, float imPart) {
        rePart_ = rePart;
        imPart_ = imPart;
    }
    ...
}
public class Main {
    public static void main(String[] args) {
        Complex comp = new Complex(5, 6);
        System.out.println(comp.rePart_); // Hiba
        System.out.println(comp); // Nem hiba
    }
}
```

Ismétlés: memóriakép




```
Position pos1;
Position pos2 = new Position(3.0f, 2.5f);
pos1 = new Position(5.0f, -1.0f);
```

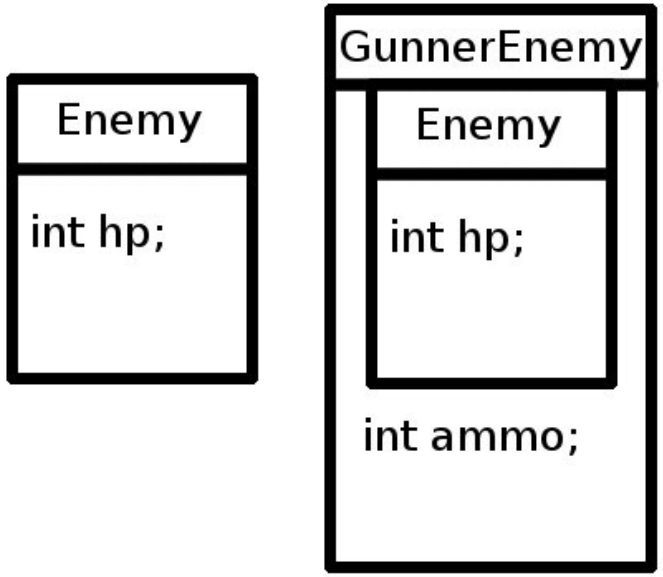
Ismétlés: statikus adattag, metódus

```
public class Pelda {
    public static int szamlalo = 0;

    public Pelda() {
        Pelda.szamlalo++;
    }
    public static String koszon() {
        return "Hello!";
    }

    public static void main(String[] args) {
        System.out.println(Pelda.koszon()) // Hello!
        Pelda pelda1 = new Pelda();
        Pelda pelda2 = new Pelda();
        System.out.println(Pelda.szamlalo); // 2
    }
}
```

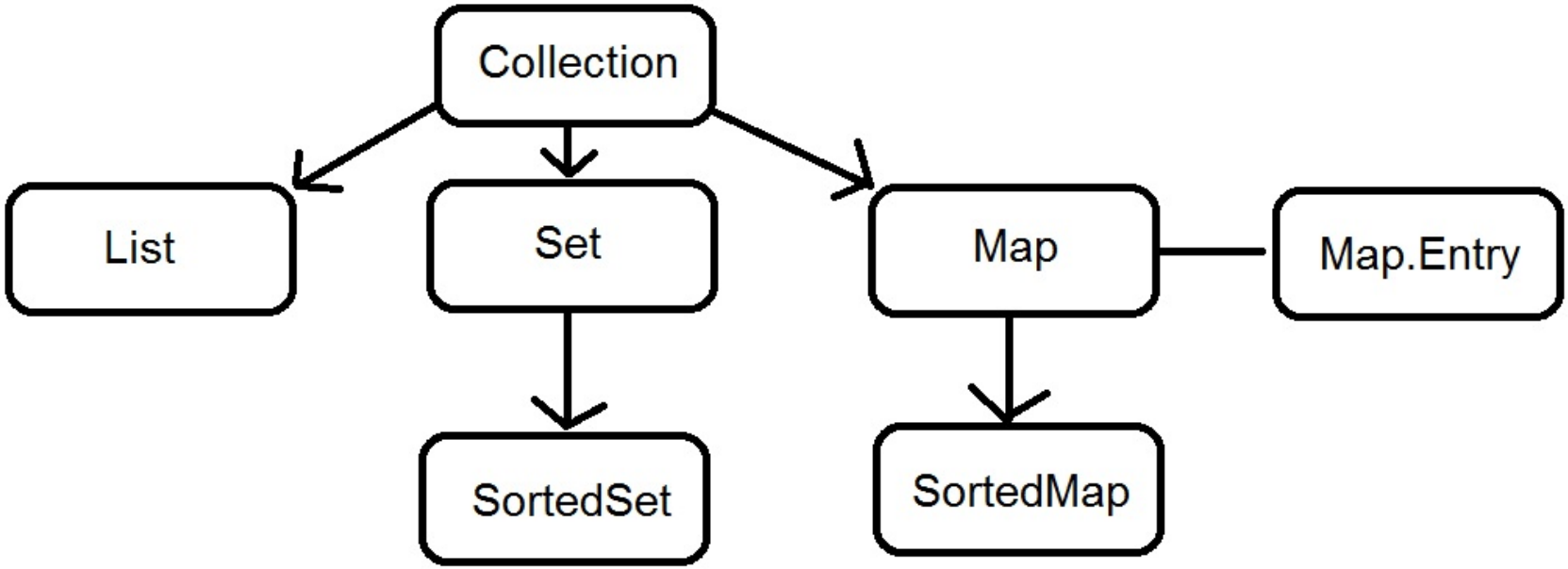
Ismétlés: öröklés, interface




```
public interface Drawable {
   public void draw();
}
...
      Drawable drawables[] = new Drawable[2];

      drawables[0] = new SpaceShip(...);
      drawables[1] = new Asteroid(...);
      for(Drawable d : drawables) {
         d.draw();
      }
...
```

Adatszerkezetek



- 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. (Dictionary)

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. Kiemelve írom a **Collection**-t, amin a metódusok meg vannak hívva, amikor nem lenne egyértelmû.

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. Kiemelve írom a **Collection**-t, amin a metódusok meg vannak hívva, amikor nem lenne egyértelmû.

- boolean add(Object o)
  - A Collection végére beteszi az adott objektumot.
- boolean addAll(Collection c)
  - A kapott Collection összes elemét beteszi a **Collection**be.
- 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.

List

- 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[]) {
      // Létrehozunk egy ArrayList-et
      ArrayList al = new ArrayList();
      System.out.println("Initial size: " + al.size());
      // hozzáadunk elemeket
      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());
    // kiírtuk a méretét, majd szépen ki is tudjuk írni:
    System.out.println("Contents: " + al);
    // Elemeket törölhetünk
    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]
```

Set

- 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();
      // hozzáadunk elemeket
      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]

Map

- 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));

      // A tárolt párok halmazát lekérjük
      Set set = hm.entrySet();
      // Majd ennek egy iterátorát
      Iterator i = set.iterator();
      // Display elements
}
```

```
        while(i.hasNext()) { // amíg van következõ elem
         Map.Entry me = (Map.Entry)i.next();
         System.out.print(me.getKey() + ": ");
         System.out.println(me.getValue());
      }
      // Hozzáadunk 1000-et Zara számlájához
      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");
      // Iterátorral kiírjuk az al elemeit
      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();
```

```
        // Változtathatjuk is az elemeket
        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+
```

Tárolt típus

- 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>();
```

Tárolt típus

- 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, mivel tudjuk, hogy ezek implementálják a korábban említett interface-eket, így a következõt is írhatjuk (és fogjuk is írni):

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

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

Kényelem

- Tanultunk már egy speciális **for** ciklust, amivel bejárhatunk pl egy tömböt.
- Ezt a szintaxist használhatjuk bármilyen **Collection**-re is:

```
List<Integer> testList = new ArrayList<Integer>();
testList.add(5);
testList.add(6);
testList.add(7);

for(Integer i : testList) {
    System.out.println(i);
}
```

- Amint látjátok primitíveket nem tudnak a **Collection**-ök tárolni, így muszáj volt az **Integer** osztályt használni.

Mapek használata

- A **Map**-ek kicsit bonyolultabban járhatók be:

```
Map<String, Integer> map = new HashMap<String, Integer>();

map.put("kutya", 5);
map.put("macska", 11);

for(Map.Entry<String, Integer> pair : map.entrySet()) {
    System.out.println(pair);
}
```

- A **Map**-nek két típust kell megadni, a kulcs típusát és a tárolt érték típusát.
- Ciklusban pedig csak a tárolt párokon tudunk végigmenni. Ezt úgy tesszük meg, hogy ennek egy halmazát (**Set**) kérjük le.
- A **Map.Entry** a tárolt pároknak a típusa.