

Java Collections



Peter Borovanský
KAI, I-18

borovan 'at' ii.fmph.uniba.sk
<http://dai.fmph.uniba.sk/courses/JAVA/>

Java Collections



dnes bude:

- podtriedy Collection
 - množiny (sets)
 - zoznamy (lists)
 - fronty (queues)
 - zobrazenia (maps) – asociativne polia, adš

Cvičenie:

- prioritný front
- HashSet, ArrayList, HashMap, ...

literatúra:

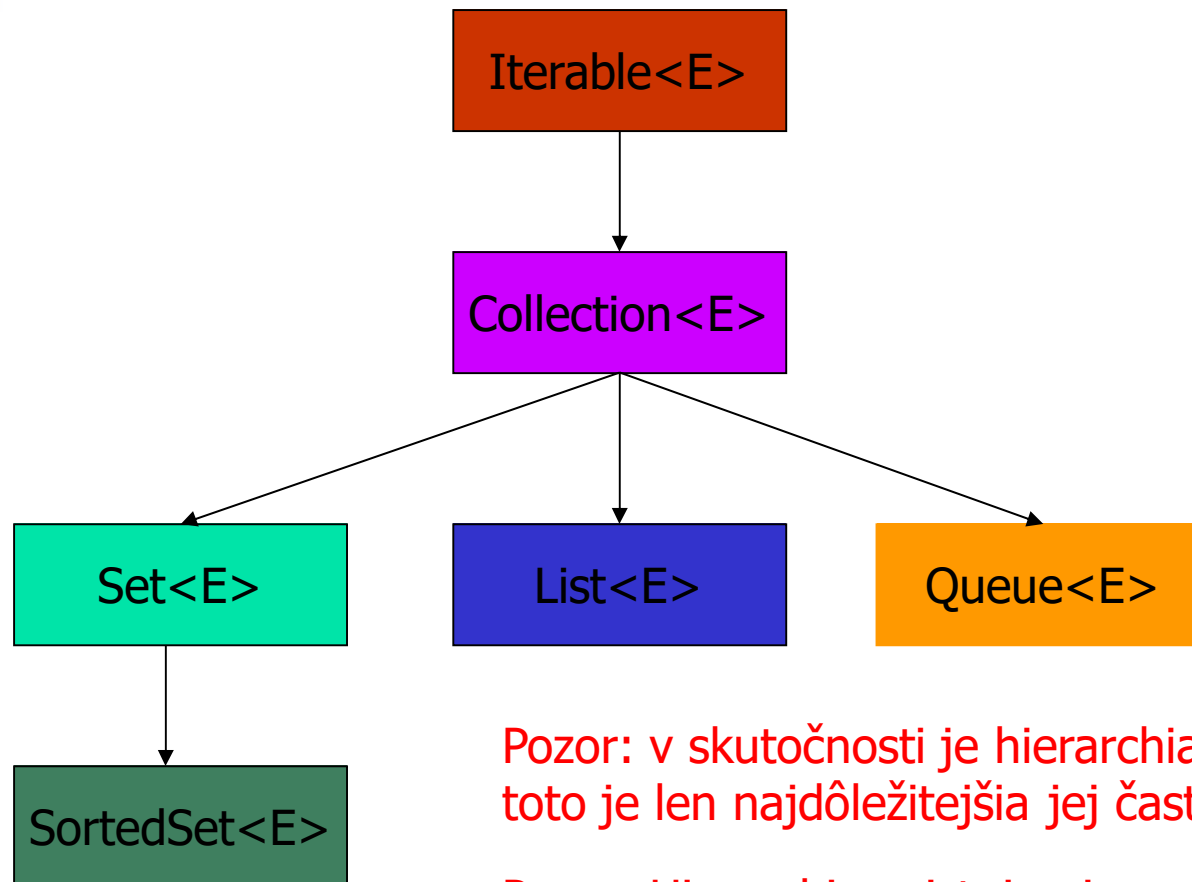
- Thinking in Java, 3rd Ed. (<http://www.ibiblio.org/pub/docs/books/eckel/TIJ-3rd-edition4.0.zip>) – 11: Collections of Objects



Java Collections

- ~~s poliami by sme si ešte dlho vystačili~~, **JAVA Collections** patria k **používaným triedam zručného programátora** - podobne ako C++ kontajnery v STL,
- Hoc ide len o knižničné triedy, budeme sa im venovať z troch pohľadov:
 1. **interface** - aký ADT definujú
 2. **implementation** - zvolená reprezentácia pre ADT
 3. **algorithm** - ako efektívne sú realizované metódy
- pre **eleganciu kódovania**: treba mať prehľad v kolekciách po kontajneroch STL nás neprekvapí, že najpoužívanéjšie z nich sú:
 1. set=množina
 2. list=zoznam
 3. queue=front
 4. map=zobrazenie
- pre **efektívnosť kódu**: treba mať predstavu o ich implementácii

Iterable interface hierarchy



Pozor: v skutočnosti je hierarchia oveľa košatejšia
toto je len najdôležitejšia jej časť

Pozor: Hierarchia existuje aj v negenerickej verzii,
neradno ich miešať (heterogénne kolekcie z Object)

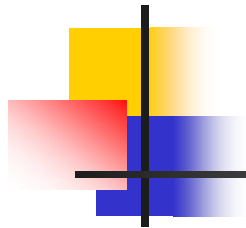
```
public interface Iterable<T> {  
    public Iterator<T> iterator();  
}
```



Iterable/Iterator interface

Iterable/Iterator interface umožňuje sekvenčný prechod ľubovoľnou collection:

```
public interface Iterator<E> {  
    boolean hasNext();           // true, ak som na poslednom prvku  
    E next();                   // chod' na ďalší prvok  
    void remove();              // vyhod' ten, na ktorom stojíš–voliteľné  
  
    // ako prejsť collection, nechať x také, že platí cond(x)  
    static <E> void filter(Collection<E> c) {  
        for (Iterator<E> it = c.iterator(); it.hasNext(); )  
            if (!cond(it.next()))           // cond je logická podmienka  
                it.remove();  
    }  
  
    static <E> void printCollection(Collection<E> c) {  
        for (Iterator<E> it = c.iterator(); it.hasNext(); )  
            System.out.println(it.next());  
    }  
}
```



Generické vs. negenerické

(homogénne vs. heterogénne)

// generická kolekcia neznámeho typu

```
static void printCollection(Collection<E> c) {  
    for (Iterator<E> it = c.iterator(); it.hasNext(); )  
        System.out.println(it.next());  
}
```

// negenerická (hetero-) kolekcia

```
static void printCollection(Collection c) {  
    for (Iterator it = c.iterator(); it.hasNext(); )  
        System.out.print(it.next());  
}
```

// cyklus for-each na homogénnych kolekciach

```
static <E> void printCollection(Collection<E> c) {  
    for (E elem : c) System.out.print(elem);  
}
```

// cyklus for-each na heterogénnych kolekciach

```
static void printCollection(Collection c) {  
    for (Object o : c) System.out.print(o);  
}
```



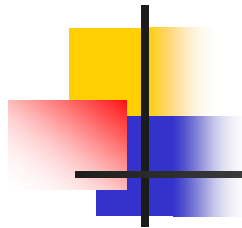
Interface Collection<E>

Spoločné minimum pre všetky triedy implementujúce Collection:

```
public interface Collection<E> extends Iterable<E> {  
    int size(); // veľkosť  
    boolean isEmpty(); // či je prázdna  
    boolean add(E element); // pridaj do nej  
    boolean contains(Object element); // nachádza sa v nej  
    boolean remove(Object element); // vyhod' prvok  
    Iterator<E> iterator(); // iterátor cez collection  
  
    ....  
    Object[] toArray(); // konverzia do poľa Object-ov  
    <T> T[] toArray(T[] a); // konverzia do poľa T[]  
}
```

Ďalšie pod-interfaces prikazujú implementovať iné partikulárne metódy:

....., Deque<E>, List<E>, Queue<E>, Set<E>, SortedSet<E>,

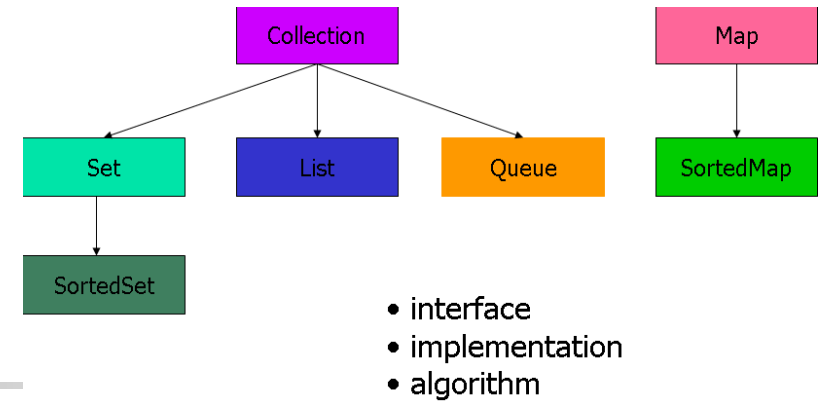


Implementácie Collection<E>

.. je ich veľa, všimnime si dôležitejšie ..

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayDeque, ArrayList, AttributeList, BeanContextServices Support,	BeanContextSupport, ConcurrentLinked Queue, ConcurrentSkipListSet, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingDeque, LinkedBlockingQueue, LinkedHashSet, LinkedList,	PriorityBlocking Queue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector
--	--	---

Implementation



Inter faces	Implementations				
	Hash table	Resizable array	Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Najčastejšia implementácia

Dôležité vedieť:

1. zobrazenie Map obsahuje páry (*key;object*) prístupné cez kľúč *key = dictionary*
2. TreeMap a TreeSet sú usporiadané (podľa kľúča), potrebujú usporiadanie na prvkoch (na kľúčoch)
3. Set a Map nemôžu obsahovať duplikáty (pre kľúč), porovnanie na rovnosť



Množina - Set

prvky sa neopakujú

implementácie:

- HashSet
- LinkedHashSet
- TreeSet - usporiad
- EnumSet

```
public interface Set<E> extends Collection<E> {  
    int size();  
    boolean isEmpty();  
    boolean add(E element);  
    boolean contains(Object element);  
    boolean remove(Object element);  
    boolean containsAll(Collection<?> c);  
    boolean addAll(Collection<? extends E> c);  
    boolean removeAll(Collection<?> c);  
  
    Iterator<E> iterator();  
    Object[] toArray();  
    <T> T[] toArray(T[] a);  
}
```

```
// pridaj  
// nachádza sa  
// vyhod'  
// podmnožina  
// zjednotenie  
// rozdiel'
```

```
// konverzia do  
// poľa
```

Príklad HashSet (Set)

- HashSet negarantuje poradie pri iterácii

- LinkedHashSet iteruje v poradí insertu prvkov

```
import java.util.HashSet;
```

```
public class HashSetDemo {  
    public static void main(String[] args) {
```

```
        HashSet<String> s = new HashSet<String>();
```

```
        for (String a : args)
```

```
            if (!s.add(a)) // nepodarilo sa pridať
```

```
                System.out.println("opakuje sa: " + a);
```

```
        System.out.println(s.size() + " rozne: " + s);
```

```
        Object[] poleObj = s.toArray();
```

```
        for(Object o:poleObj) System.out.print(o);
```

```
        String[] poleStr = s.toArray(new String[0]);
```

```
        for(String str:poleStr) System.out.print(str);
```

```
    }  
}
```

Konverzia do poľa

Podhodím mu typ poľa aby vedel...

java HashSetDemo a b b a

opakuje sa: b

opakuje sa: a

2 rozne: [a, b]

abab

Usporiadaná množina – TreeSet iteruje podľa usporiadania


SortedSet

prvky sa neopakujú a navyše sú usporiadané

okrem toho, čo ponúka Set<E> dostaneme:

```
public interface SortedSet<E> extends Set<E> {  
    SortedSet<E> subSet(E from, E to);           // vykusne podmnožinu  
                                                    // prvkov od  $\geq$  from a  $<$  to  
    SortedSet<E> headSet(E toElement);          // podmnožina prvkov  
                                                    // od začiatku až po toElement  
    SortedSet<E> tailSet(E fromElement);         // podmnožina prvkov  
                                                    // od fromElement až po koniec  
    E first();                                    // prvý  
    E last();                                    // posledný prvok usp.množiny  
}  
headSet(to)  
subSet(from, to)  
{ first < ... < from < ... < to < ... < last }  
tailSet(from)
```

Príklad TreeSet (SortedSet)



```
import java.util.TreeSet;

public class TreeSetDemo {
    public static void main(String[] args) {

        List<String> list = Arrays.asList( // rozdelí reťazec na slová do poľa, z ktorého
            "jedna dva tri styri pat sest sedem osem devat".split(" ")); //vyrobí List

        TreeSet<String> sortedSet = new TreeSet<String>(list); //vyrobí SortedSet
        System.out.println(sortedSet);
            // [devat, dva, jedna, osem, pat, sedem, sest, styri, tri]

        String low = sortedSet.first(), high = sortedSet.last(); // devat, tri
        System.out.println(sortedSet.subSet("osem", "sest")); // [osem, pat, sedem]
        System.out.println(sortedSet.headSet("sest"));
            // [devat, dva, jedna, osem, pat, sedem]
        System.out.println(sortedSet.tailSet("osem"));
            // [osem, pat, sedem, sest, styri, tri]
    }
}
```



Zoznam - List

implementácie:

- ArrayList
- LinkedList
- Vector
- Stack

prvky sa môžu opakovať a majú svoje poradie, resp. usporiadanie
ListIterator okrem next()/hasNext(), pozná aj previous()/hasPrevious()

```
public interface List<E> extends Collection<E> {
```

```
    E get(int index);           // priamy prístup cez index - getter
    E set(int index, E element); // priamy prístup cez index - setter
    boolean add(E element);     // pridaj
    void add(int index, E element); // pridaj na pozíciu
    E remove(int index);        // vyhod'
    boolean addAll(int index, Collection<? extends E> c); // vsuň celú coll.
    int indexOf(Object o);       // vyhľadávanie, vráti index, ak nájde
    int lastIndexOf(Object o);
    ListIterator<E> listIterator(); // bohatší iterátor, vie ísť aj od zadu
    ListIterator<E> listIterator(int index); // iteruj od indexu
    List<E> subList(int from, int to); // podzoznam
```

```
}
```

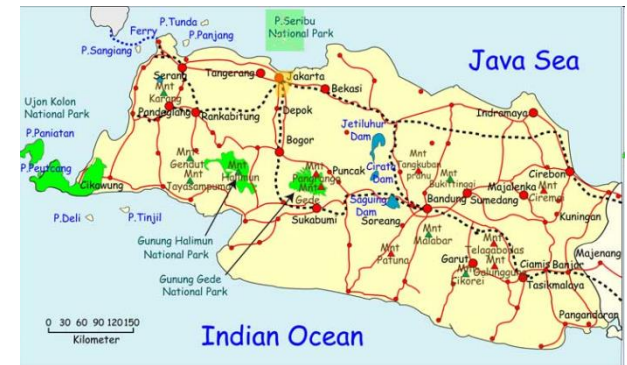
Príklad ArrayList (List)

```
import java.util.*;
```

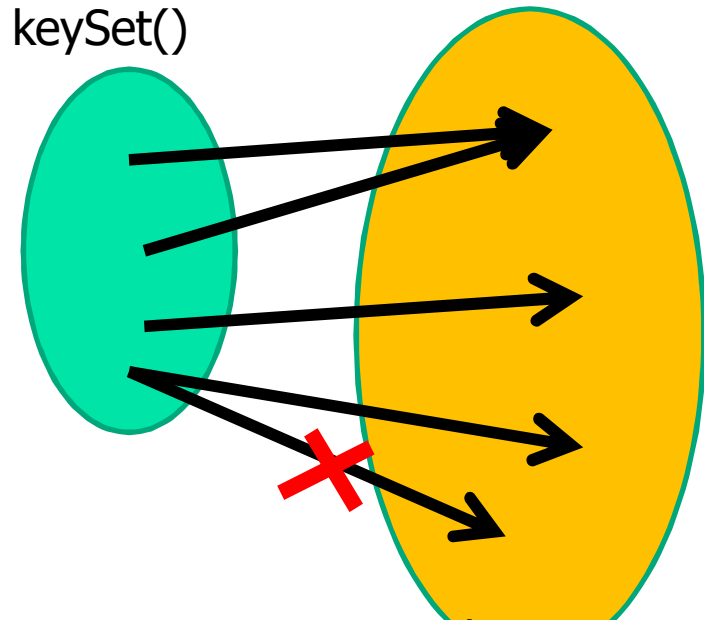
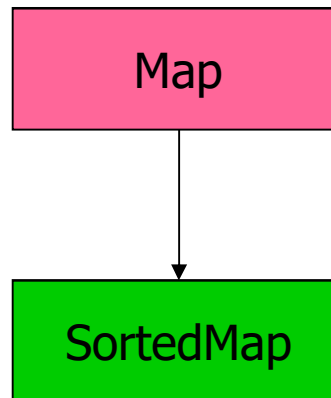
```
public class ArrayListDemo {  
    public static void main(String[] args) {  
        String[] p = {"a","b","c","d"};  
        ArrayList<String> s=new ArrayList<String>(); // vytvor prazdny zoznam  
        for (String a : p) s.add(a);                // nasyp doň prvý poľa p  
        for (Iterator<String> it = s.iterator(); it.hasNext(); )  
            System.out.println(it.next());           // vypíš zoznam, abcd  
        s.set(1,"99"); s.remove(2);                  // prepíš 1. "99", vyhod' 2., a99d  
  
        for (ListIterator<String> it = s.listIterator(s.size()); it.hasPrevious(); )  
            System.out.println(it.previous());        // vypíš zoznam opačne d99a  
  
        HashSet<String> hs = new HashSet<String>();  
        hs.add("A"); hs.add("B");                     // množina [A, B]  
        s.addAll(2,hs);                                // vsunutá množina[a, 99, A, B, d]  
    }  
}
```

Map interface

(Map je zobrazenie, nie mapa)
(map je dictionary)



Kľúče
definičný obor
keySet()



Hodnoty
Obor hodnôt
values()



Interface Map<K,V>

implementácie:

- HashMap
- LinkedHashMap
- EnumMap
- TreeMap

```
public interface Map<K,V> {
```

Zobrazenie:K->V

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

m[key]=value

// pridaj dvojicu [key;value] do zobrazenia

// ak už key->value', tak prepíš

```
V get(Object key);
```

// nájdí obraz pre key, ak neexistuje->null

```
V remove(Object key);
```

m[key]

```
boolean containsKey(Object key); // patrí do definičného oboru
```

```
boolean containsValue(Object value); // patrí do oboru hodnôt
```

```
int size();
```

```
boolean isEmpty();
```

```
....
```

```
// Collection Views
```

```
public Set<K> keySet();
```

// definičný obor, množina kľúčov

```
public Collection<V> values();
```

// obor hodnôt, nemusí byť množina hodnôt

```
}
```

Príklad HashMap (Map)



```
import java.util.HashMap;
```

```
public class HaspMapDemo {  
    public static void main(String[] args) {
```

Zobrazenie:String->Integer

```
        HashMap<String, Integer> m = new HashMap<String, Integer>();  
        for (String a : args) {  
            Integer freq = m.get(a);  
            m.put(a, (freq == null) ? 1 :  
                freq + 1);  
        }  
        System.out.println(m);  
    }  
}
```

```
java HaspMapDemo x d o k o l a o k o l o k o l a
```

```
{a=2, d=1, x=1, k=3, l=3, o=6}
```



Ako prejsť cez Map

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

```
// z predošlého príkladu
```

```
{a=2, d=1, x=1, k=3, l=3, o=6}
```

- Ľahší spôsob – cez definičný obor, cez kľúče

```
for(String key : m.keySet())
```

```
    System.out.println "[" + key + "]=" + m.get(key));
```

- Ťažší spôsob – iterátorom

```
for(
```

```
    Iterator<Map.Entry<String, Integer>> it =
```

```
        m.entrySet().iterator());
```

```
    it.hasNext(); ) {
```

```
        Map.Entry<String, Integer> item = it.next();
```

```
        System.out.println "[" + item.getKey() + "]=" +  
            item.getValue());
```

```
}
```

```
[a]=2  
[d]=1  
[x]=1  
[k]=3  
[l]=3  
[o]=6
```

```
[a]=2  
[d]=1  
[x]=1  
[k]=3  
[l]=3  
[o]=6
```



Ak to bude TreeMap

(bude to utriedené podľa kľúča)

```
TreeMap<String, Integer> m = new TreeMap<String, Integer>();  
// z predošlého príkladu
```

■ Ľahší spôsob

```
for(String key : m.keySet())  
    System.out.println "[" + key + "]=" + m.get(key));
```

■ Ťažší spôsob

```
for(  
    Iterator<Map.Entry<String, Integer>> it =  
        m.entrySet().iterator();  
    it.hasNext(); ) {  
    Map.Entry<String, Integer> item = it.next();  
    System.out.println "[" + item.getKey() + "]=" +  
        item.getValue();  
}
```

{a=2, d=1, k=3, l=3, o=6, x=1}

[a]=2
[d]=1
[k]=3
[l]=3
[o]=6
[x]=1

[a]=2
[d]=1
[k]=3
[l]=3
[o]=6
[x]=1

TreeMap (Map)

```
import java.util.TreeMap;
```

```
public class CountryCapitals {  
    public static final String[][] AFRICA = {  
        {"ALGERIA", "Algiers"},  
        {"ANGOLA", "Luanda"},  
        {"BENIN", "Porto-Novo"},  
        {"BOTSWANA", "Gaberone"},  
        {"BURKINA FASO", "Ouagadougou"},  
    }
```

```
    public static TreeMap<String,String> getTreeMap(String[][] p) {  
        TreeMap<String,String> tmp = new TreeMap();  
        for(int i=0; i<p.length; i++)  
            tmp.put(p[i][0], p[i][1]);  
        return tmp;  
    }  
    public static void main(String[] args) {  
        TreeMap<String,String> europe = getTreeMap(CountryCapitals.EUROPE);  
        TreeMap<String,String> america = getTreeMap(CountryCapitals.AMERICA);
```

```
        System.out.println(europe);           // {ALBANIA=Tirana, ANDORRA=Andorra la Vella, ARMENIA=...  
        System.out.println(europe.keySet()); // [ALBANIA, ANDORRA, ARMENIA, AUSTRIA, ...  
        System.out.println(europe.values()); // [Tirana, Andorra la Vella, ...  
        europe.putAll(america);  
        System.out.println(europe);           // {ALBANIA=Tirana, ..., ARGENTINA=Buenos Aires,
```



TreeMap (Map)

(inverzia zobrazenia)

Pre ilustráciu práce so štruktúrou TreeMap vytvoríme inverziu zobrazenia (hl.mesto->štát)

```
TreeMap<String,String> inverseEurope = new TreeMap();
```

```
for(String state : europe.keySet())
```

```
    inverseEurope.put(europe.get(state),state);
```

```
System.out.println(inverseEurope);
```

```
{... Belgrade=SERBIA, Berlin=GERMANY, Berne=SWITZERLAND,  
    Bratislava=SLOVAKIA,...
```



TreeMap (Map)

(skladanie zobrazení)

// skladanie zobrazení (hl.mesto->štát) x (štát->prezident) = (hl.mesto->prezident)

inverseEurope **europePresidents**

TreeMap<String,String> **sefHlavnehoMesta** = **new** TreeMap();

```
for(String capital : inverseEurope.keySet()) {  
    String state = inverseEurope.get(capital);  
    if (state != null) {  
        String president = europePresidents.get(state);  
        if (president != null)  
            sefHlavnehoMesta.put(capital,president);  
    }  
}  
System.out.println(sefHlavnehoMesta);  
{Bratislava=Kiska, Moscow=Putin, Prague=Zeman}
```



TreeMap (Map)

(inverzia zobrazenia – oveľa ťažkopádnejšie)

Pre ilustráciu práce so štruktúrou TreeMap vytvoríme inverziu zobrazenia (hl.mesto->štát)

```
TreeMap<String,String> inverseEurope = new TreeMap();

for(Iterator<Map.Entry<String,String>> it = europe.entrySet().iterator();
    it.hasNext(); ) {
    Map.Entry<String,String> item = it.next(); // prechádzam prvky
                                              // pôvodného zobrazenia
    inverseEurope.put(item.getValue(),item.getKey()); // čo bolo kľúč je
                                                       // hodnota a naopak
}
System.out.println(inverseEurope);
{... Belgrade=SERBIA, Berlin=GERMANY, Berne=SWITZERLAND,
    Bratislava=SLOVAKIA,...
```




TreeMap (Map)

(skladanie zobrazení – oveľa ťažkopádnejšie)

// skladanie zobrazení (hl.mesto->štát) x (štát->prezident) = (hl.mesto->prezident)

```
TreeMap<String,String> sefHlavnehoMesta = new TreeMap();
```

```
for(Iterator<Map.Entry<String,String>> it= inverseEurope.entrySet().iterator();  
    it.hasNext();){  
    Map.Entry<String,String> item = it.next(); // prechádzame jedno zobrazenie  
    String president = europePresidents.get(item.getValue()); // hodnotu zobrazíme v  
                                                                // druhom zobrazení  
    if (president != null) // ak v druhom má hodnotu, tak  
        sefHlavnehoMesta.put(item.getKey(),president);  
                                                                // pôvodný kľúč a zobrazenú hodnotu  
                                                                // pridáme do zloženého zobrazenia  
}  
System.out.println(sefHlavnehoMesta);  
{Bratislava=Kiska, Moscow=Putin, Prague=Zeman}
```