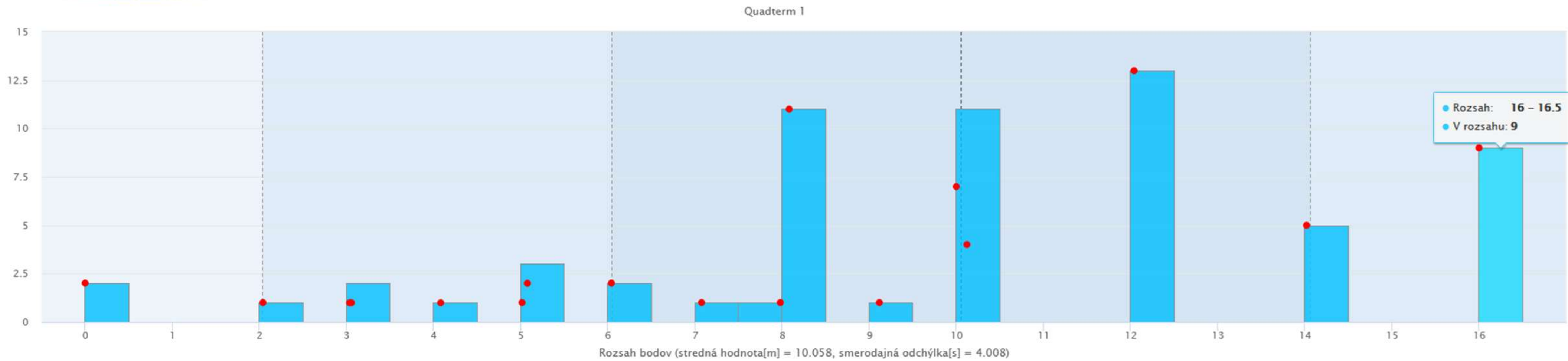
Java Collections

Peter Borovanský
KAI, I-18

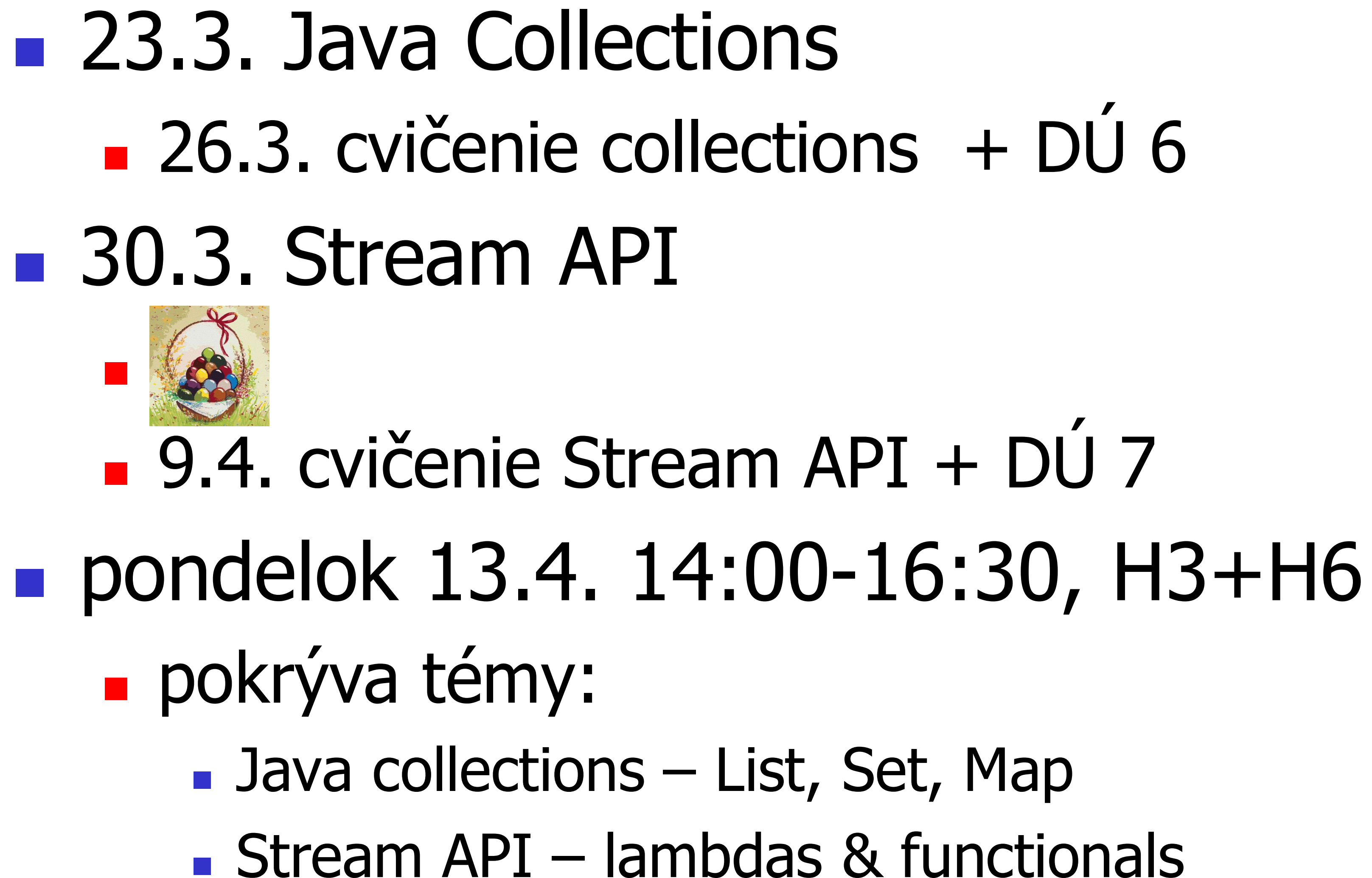
borovan 'at' ii.fmph.uniba.sk
http://dai.fmph.uniba.sk/courses/JAVA/

Quadterm-1



- priemer 10.01/16
- počet 68

Harmonogram

- 23.3. Java Collections
  - 26.3. cvičenie collections + DÚ 6
- 30.3. Stream API
  - 
  - 9.4. cvičenie Stream API + DÚ 7
- pondelok 13.4. 14:00-16:30, H3+H6
  - pokrýva témy:
    - Java collections – List, Set, Map
    - Stream API – lambdas & functionals

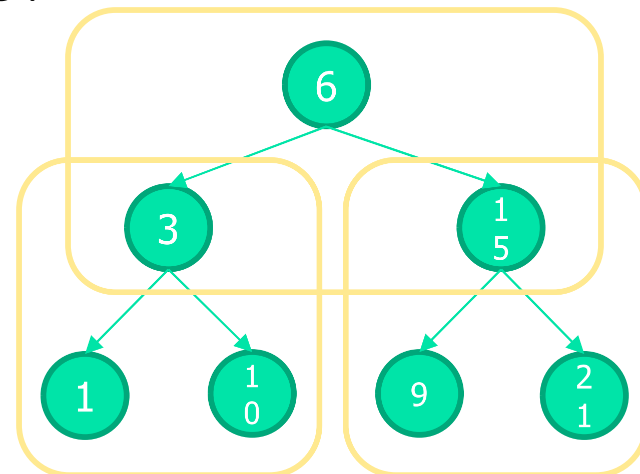
BST

z dielne majstrov :)

```
abstract class Tree {
   abstract boolean isBST();
   ...
class Node extends Tree {
  @Override
   boolean isBST() {
      if (left == null && right == null) return true;
      if (left != null) {
         if (data <= left.root()) return false;
         if (!left.isBST()) return false;
      }
      if (right != null) {
         if (data >= right.root()) return false;
         if (!right.isBST()) return false;
      }
      return true;
}  }

class Leaf extends Tree {
  @Override
   boolean isBST() {
      return true;
}  }
```

```
Tree c3 = new Node(new Leaf(1), 3, new Leaf(10));
Tree c4 = new Node(new Leaf(9), 15, new Leaf(21));
Tree c5 = new Node(c3, 6, c4);

@Override
boolean isBST() {
  return
    (left == null || left.root() < data     && left.isBST())
     &&
    (right == null || right.root() > data && right.isBST());
}
```

Zlé riešenie, a vieme napísať fungujúce lineárne riešenie ?



BST.java

BST v štýle J17

Sealed class/interface

Zapečatená (sealed) trieda/interface určuje, ktoré **jediné** podtriedy trieda môže mať, resp. ktoré triedy **jediné** implementujú zapečatený interface **(a žiadne iné) – idea zapečatenosti**

```
sealed interface Tree permits Node, Leaf {
  boolean isBST();
}

record Node(Tree left, int value, Tree right) implements Tree {
   @Override
   public boolean isBST() {  // bez zmeny
      return     (left == null || left.root() < value && left.isBST())
              && (right == null || right.root() > value && right.isBST());
      }
}

record Leaf(int value) implements Tree {
  @Override
   public boolean isBST() {
         return true;
}  }
```

BST1.java

BST v štýle J17

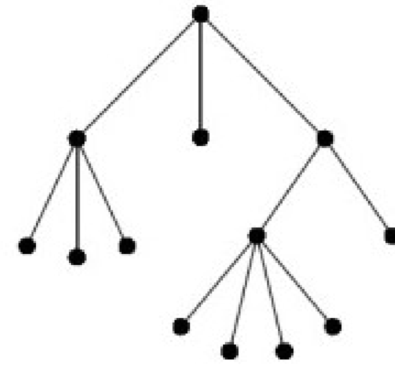
patterns

```
static int velkost1(Tree t) {
   return
      switch (t) {        // switch je príkaz !!!
         case Node n ->
            1 +
            ((n.left() == null) ? 0:velkost1(n.left())) +
            ((n.right() == null) ? 0:velkost1(n.right()));
         case Leaf l -> 1;

         // no default... lebo Tree je zapečatený

      };
   }
```

```
// pred Java 17
static int velkost2(Tree t) {
   if (t instanceof Node) {
      Node n = (Node) t;
      return
        1 +
        ((n.left() == null)?0:velkost1(n.left())) +
        ((n.right() == null)?0:velkost1(n.right()));
   } else if (t instanceof Leaf) {
      Leaf l = (Leaf)t;
      return 1;
   } else {
      return 999;
   }
}
```

BST1.java

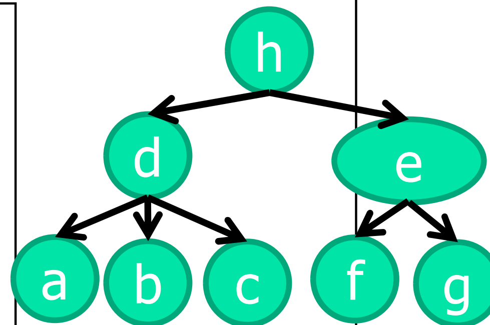
Všeobecný strom



```
class Node<T extends Comparable<T>> implements Comparable<Node<T>>  {
   private T           value;
   private List<Node<T>> sons;
```

- T extends Comparable<T> - znamená, že predpokladáme porovnávanie na type T
- implements Comparable<Node<T>> - znamená, že chceme porovnávať celé stromy
  - a preto musíme definovať
  - @Override
    public int compareTo(Node<T> o) { ... }
- príklad rekurzie cez strom, inicializácie:

```
public int size() {
   int count = 1;
   for (Node<T> son : sons)
      if (son != null)
         count += son.size();
   return count;
}
```




```
Node<String> tree = new Node("h",
      List.of(
         new Node("d",
            List.of(
               new Node("a", null),
               new Node("b", null),
               new Node("c", null))
         ),
         new Node("e",
            List.of(
               new Node("f", null),
               new Node("g", null))
         )));
```

Java Collections

- Neberte si kľúčiky od veľkej miešačky, kým neviete robiť s malou lopatou

- všetky príklady Quadtermu boli riešiteľné bez Listu, Set či Map

Sú collections lepšie ?

(a môžeme ich už používať?)

```
public static void bubleSortuj(int[] a) {
    for (int i = 0; i < a.length ; i++) {
      for (int j = a.length-1; j>i ; j--) {
        if (a[j-1] > a[j]) {
            int temp = a[j];
            a[j] = a[j-1];
            a[j-1] = temp;
        }
      }
    }
}
```

10^4 - elapsed time:141 milis.

10^5 - elapsed time:14s.

10^6- elapsed time: ???

```
public static void
          bubleSortuj(ArrayList<Integer> a) {
      for (int i = 0; i < a.size() ; i++) {
        for (int j = a.size()-1; j>i ; j--) {
          if (a.get(j-1) > a.get(j)) {
            Integer temp = a.get(j);
            a.set(j,a.get(j-1));
            a.set(j-1,temp);
          }
        }
      }
}
```

10^5 - elapsed time:33 s. – 2.35x pomalšie

Súbor: **SortObama1.java**, **SortObama2.java**

Je Python lepší ?

```
import random
import datetime
def bubbleSort(alist):
    for passnum in range(len(alist)-1,0,-1):
        for i in range(passnum):
            if alist[i]>alist[i+1]:
                temp = alist[i]
                alist[i] = alist[i+1]
                alist[i+1] = temp
alist = random.sample(range(100000000), 10000)
start = datetime.datetime.now()
bubbleSort(alist)
now = datetime.datetime.now()
print(alist)
print(now-start)
```

Verzia ADŠ (http://struct.input.sk/07.html#triedenia):

```
def bubble_sort(pole):
    for i in range(1, len(pole)):
        for j in range(len(pole)-i):
            if pole[j] > pole[j+1]:
                pole[j], pole[j+1] = pole[j+1], pole[j]
```

10^4 - elapsed time:15 s. - 106x pomalšie...

10^5 - elapsed time: 28m05s - 120x pomalšie...

http://interactivepython.org/courselib/static/pythonds/SortSearch/TheBubbleSort.html

QuickSort

	python quicksort	.sort() builtin
10^6	7.81s	0.88s
10^6	102.3s	16.2s

v Jave:

10^5 - elapsed time: 23ms. (608x)

10^6- elapsed time: 120ms

10^7- elapsed time: 1.2s

10^8- elapsed time: 12.31s

10^9- elapsed time: 123.8s

builtin, Arrays.*sort:*

10^5 - elapsed time:39ms.

10^6- elapsed time: 129ms

10^7- elapsed time: 1.02s

10^8- elapsed time: 10.5s

10^9- elapsed time: 101.5s

http://www.vogella.com/tutorials/JavaAlgorithmsQuicksort/article.html

Súbor: QuickSort.java, QuickSortTest.java

Java Collections

dnes bude:

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

Slajd < Java 8
(všade)

Slajd = Java 8

Slajd = Java 9

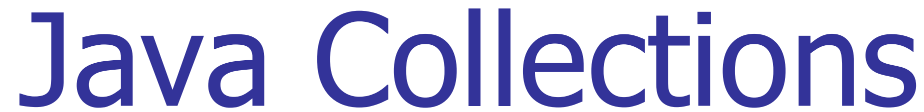
Slajd = Java 10

Cvičenie:

- 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

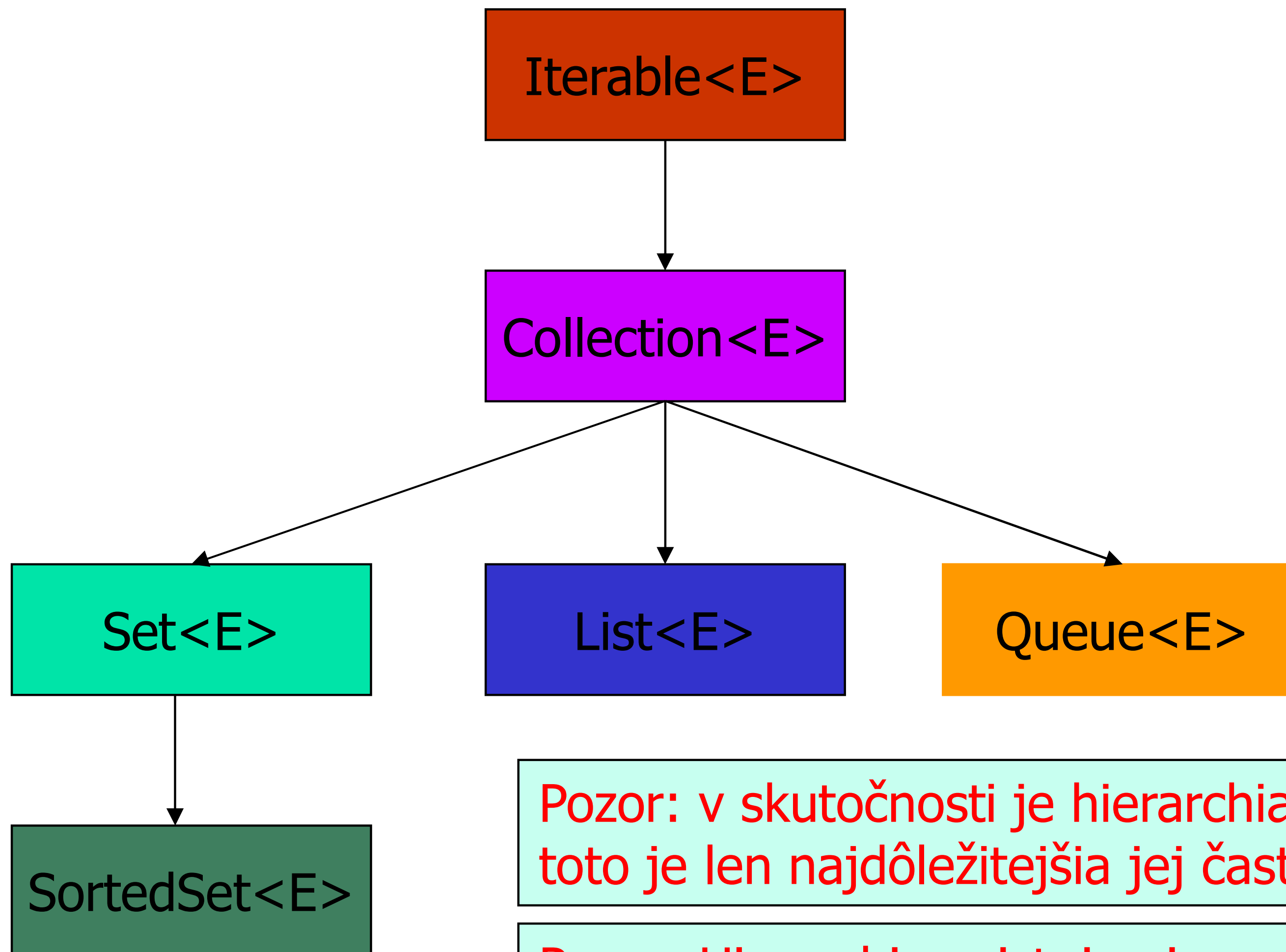
- ~~s poliami by sme si ešte dlho vystačili~~, JAVA Collections patria k používaným triedam zručného programátora - podobne ako C++ kontajnery v STL,

- 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 kolekciach
  po kontajneroch STL nás neprekvapí, že najpoužívanejšie z nich sú:
  1. set=množina
  2. list=zoznam
  3. queue=front
  4. map=zobrazenie

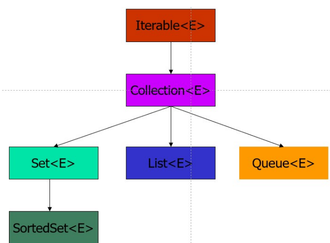
- pre efektívnosť kódu: treba mať predstavu o ich implementácii

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();                   // choď na ďalší prvok
    void remove();              // vyhoď ten, na ktorom stojíš–
                                       voliteľné
```

// ako prejsť collection, nechať x také, že platí cond(x)

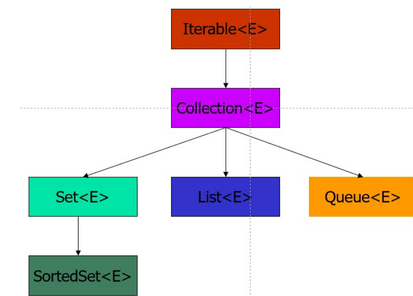
```
static <E> void filter(Collection<E> c) {
    for (Iterator<E> it = c.iterator(); it.hasNext(); )
        if (!cond(it.next()))       // cond je logická podmienka
            it.remove();
}

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 homogénna kolekcia typu E
static void printCollection(Collection<E> c) {
    for (Iterator<E> it = c.iterator(); it.hasNext(); )
        System.out.println(it.next());     // typu E
}
                      // negenerická hetero (Any/Object) kolekcia
static void printCollection(Collection c) {
   for (Iterator it = c.iterator(); it.hasNext(); )
        System.out.print(it.next());       // typu Object
}
                     // 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 je podinterface Iterable

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);        // vyhoď prvok
    Iterator<E> iterator();                // iterátor cez collection
    ….
    Object[] toArray();                 // konverzia do poľa Objectov
    <T> T[] toArray(T[] a);                // konverzia do poľa T[]
}
```

Ďalšie pod-interfaces prikazujú implementovať iné partikulárne metódy:

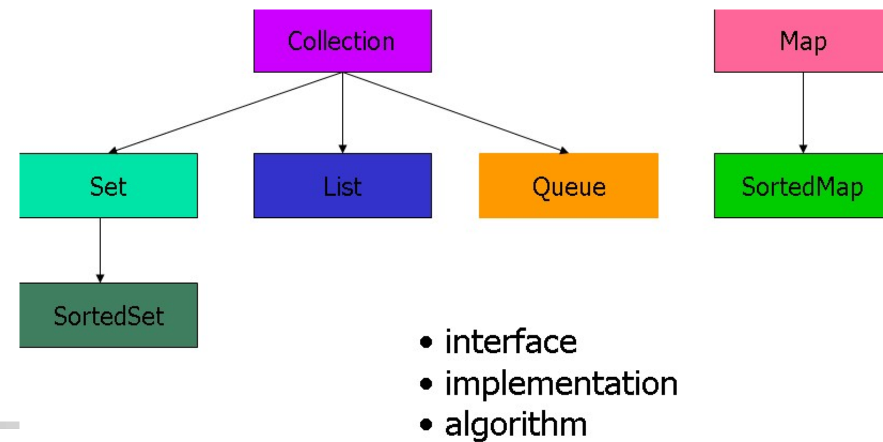
……, Deque<E>, List<E>, Queue<E>, Set<E>, SortedSet<E>, ……

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

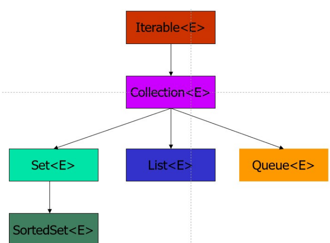


Inter faces	Implementations				
	Hash table	Resizable array	Tree	Linked List	Hash table + Linked list
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue		ArrayList		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Najčastejšia implementácia

Dôležité vedieť:

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



Množina - Set

implementácie:
- HashSet
- LinkedHashSet
- TreeSet - usporiad
- EnumSet

prvky sa neopakujú

```
public interface Set<E> extends Collection<E> {
   int size();
   boolean isEmpty();
   boolean add(E element);                      // pridaj
   boolean contains(Object element);            // nachádza sa
   boolean remove(Object element);              // vyhoď
   boolean containsAll(Collection<?> c);        // podmnožina
   boolean addAll(Collection<? extends E> c);// zjednotenie
   boolean removeAll(Collection<?> c);          // rozdiel

   Iterator<E> iterator();
   Object[] toArray();                          // konverzia do
   <T> T[] toArray(T[] a);                      // poľa
}
```

Príklad HashSet (Set)

- HashSet negarantuje poradie pri iterácii
- LinkedHashSet iteruje v poradí vkladania prvkov

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

```
                          // >= Java 1.5, zaviedla generics
Set<String> s = new HashSet<String>();
                       // >= Java 1.7, zaviedla diamond operator <>
Set<String> s = new HashSet<>();
Set<String> s = new HashSet<~>(); // špecialitka IntelliJ
Set<String> s = new LinkedHashSet<>();
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);
```

(The line `Set<String> s = new HashSet<~>(); // špecialitka IntelliJ` is struck through.)

Konverzia do poľa

Podhodím mu typ poľa, aby vedel...

HashSetDemo a b b a
opakuje sa: b
opakuje sa: a
2 rozne: [a, b]
abab

Súbor: HashSetDemo.java

< Java 8

Java 8

Java 9

Java 10

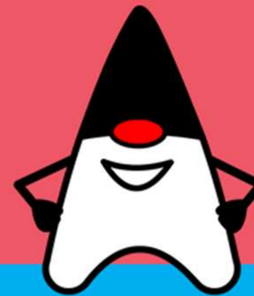
Množinová konštanta

```
Set<Integer> s = new HashSet<>();
s.add(1); s.add(2); s.add(3);
```

```
Set<String> strs = new HashSet<>(          // <> Java 7
        Arrays.asList("Java", "Kawa"));
        alebo…
        Arrays.asList(new String[]{"Java", "Kawa"}));
```

```
Set<Integer> r = Set.of(1,2,3);
Set<String> r = Set.of("Java", "Kawa");
```

```
var q = Set.of(true, false);

// Set<Set<Integer>> powerSet matematicky {{},{0},{1},{0,1}}
var powerSet = Set.of(
   Set.of(), Set.of(0), Set.of(1), Set.of(0,1) );
```



Java 10

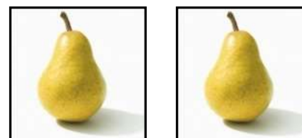
Implicitná typová deklarácia-typ premennej si domyslí s inicializačnej hodnoty

```
        var a = 0;
        var h = new Hruska();
        var pole = new String[10];
        var list = new ArrayList<String>();
        var map = new HashMap<String, String>();
class Hruska { ... }
```

Nič to ale nemení na fakte, že Java zostáva staticky typovaná
nikdy z toho nebude JavaScript, `var`-podobnosť je čisto náhodná...

Množina definovaných objektov

```
Hruska h1 = new Hruska(1);
Hruska h2 = new Hruska(1);
   Set<Hruska> hrusky = new HashSet<>(Arrays.asList(h1, h2));
   "size = " + hrusky.size()
   "== " + (h1 == h2)
   ".equals " + (h1.equals(h2))
```



```
size    -> 2
==      -> false
.equals-> false
```

Asi Hruske chýba náš vlastný .equals – lebo Object.equals je porovná referencie

```
class Hruska {
   int velkost;
   ...
   @Override
   public boolean equals(Object obj) {
     return
        (obj instanceof Hruska)?(velkost==((Hruska) obj).velkost)
                                        :false;
   }
```

```
size    -> 2
==      -> false
.equals-> true
```

Množina definovaných objektov

WAR IS PEACE.
FREEDOM IS SLAVERY.
IGNORANCE IS STRENGTH.
1984

Asi Hruske chýba náš vlastný .hashCode, ale ako napísať hašhovaciu funkciu ?

```
class Hruska {
    int velkost;
    ...
    @Override
    public int hashCode() {
        return 1984; //Orwellova konštanta ☺
    }
    @Override
    public int hashCode() {
         return velkost;
         return 17*velkost;
    }
    @Override
    public int hashCode() {
         return super.hashCode();
    }
```

```
size    -> 1
==      -> false
.equals-> true
```

```
size    -> 1
==      -> false
.equals-> true
```

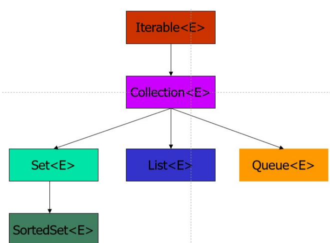
```
size    -> 2
==      -> false
.equals-> true
```

Ako teda pracuje HashSet

- žiadne duplikáty

`Set.add(key), contains(key)`, ...

- interne volá **`hashCode(key)`**
- všetky prvky s rovnakým **`hashCode`** sú v jednom (!) spájanom zozname,
  - to vysvetluje, že **`return super.hashCode();`** lebo sú všetky rôzne
- prebehne tento spájaný zoznam a porovnáva objekty pomocou **`.equals()`**
  - preto konštanta 1984 degraduje **`HashSet`** na **`LinkedList`**...
- Analogicky bude fungovať **`HashMap`** alias dictionary



Usporiadaná množina - SortedSet

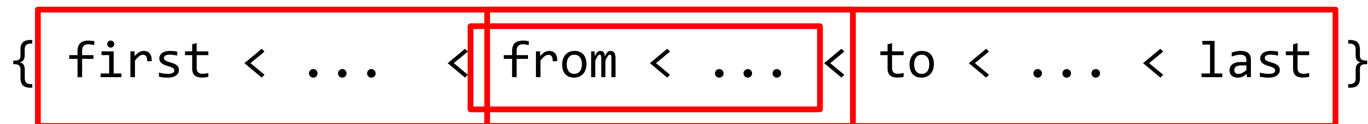
- TreeSet iteruje podľa usporiadania

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 >= 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)
```




```
tailSet(from)
```

Príklad TreeSet (SortedSet)

```
import java.util.TreeSet;

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

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

String low = sortedSet.first(), // devat
String high = sortedSet.last(); // tri

sortedSet.subSet("osem", "sest") // [osem, pat, sedem]
sortedSet.headSet("sest")    // [devat, dva, jedna, osem, pat, sedem]
sortedSet.tailSet("osem")    // [osem, pat, sedem, sest, styri, tri]
```

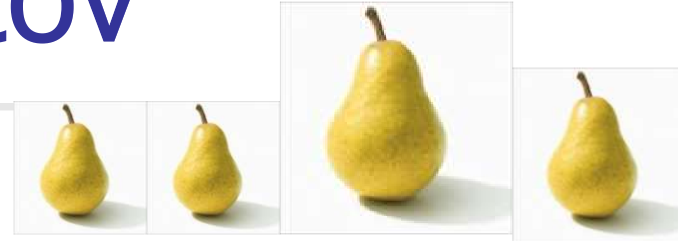
Súbor:[TreeSetDemo.java]

Usporiadana množina definovaných objektov

< Java 8

Java 8

Java 9



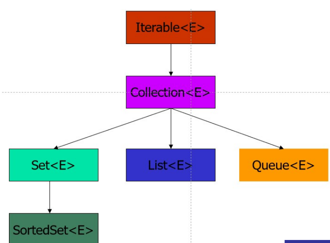
```
Hruska h1 = new Hruska(1), h2 = new Hruska(1);
Hruska h3 = new Hruska(4), h4 = new Hruska(3);
Set<Hruska> hrusky = new TreeSet<>(Arrays.asList(h4,h1,h2,h3));
Set<Hruska> hrusky = new TreeSet<>(Set.of(h4,h1,h2,h3));
```

Hruska cannot be cast to java.base/java.lang.Comparable

Asi Hruske chýba náš `implements Comparable<Hruska>` a `.compareTo()`

```
class Hruska implements Comparable<Hruska> {
   int velkost;
   ...
   @Override
   public int compareTo(Hruska o) {
    return (velkost<o.velkost)?-1:(velkost==o.velkost)?0:1;      ☹
    return new Integer(velkost).compareTo(new Integer(o.velkost)); 😐
    return Integer.compare(velkost, o.velkost);                  ☺
   }
```

```
[Hruska{velkost=1}, Hruska{velkost=3}, Hruska{velkost=4}]
```



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

< Java 8

Java 8

Java 9

Príklad ArrayList (List)

```
import java.util.*;

String[] p = {"a","b","c","d"};
ArrayList <String> s = new ArrayList<>(); // prázdny zoznam
for (String a : p)  s.add(a);           // nasyp doň prvy poľa p
```

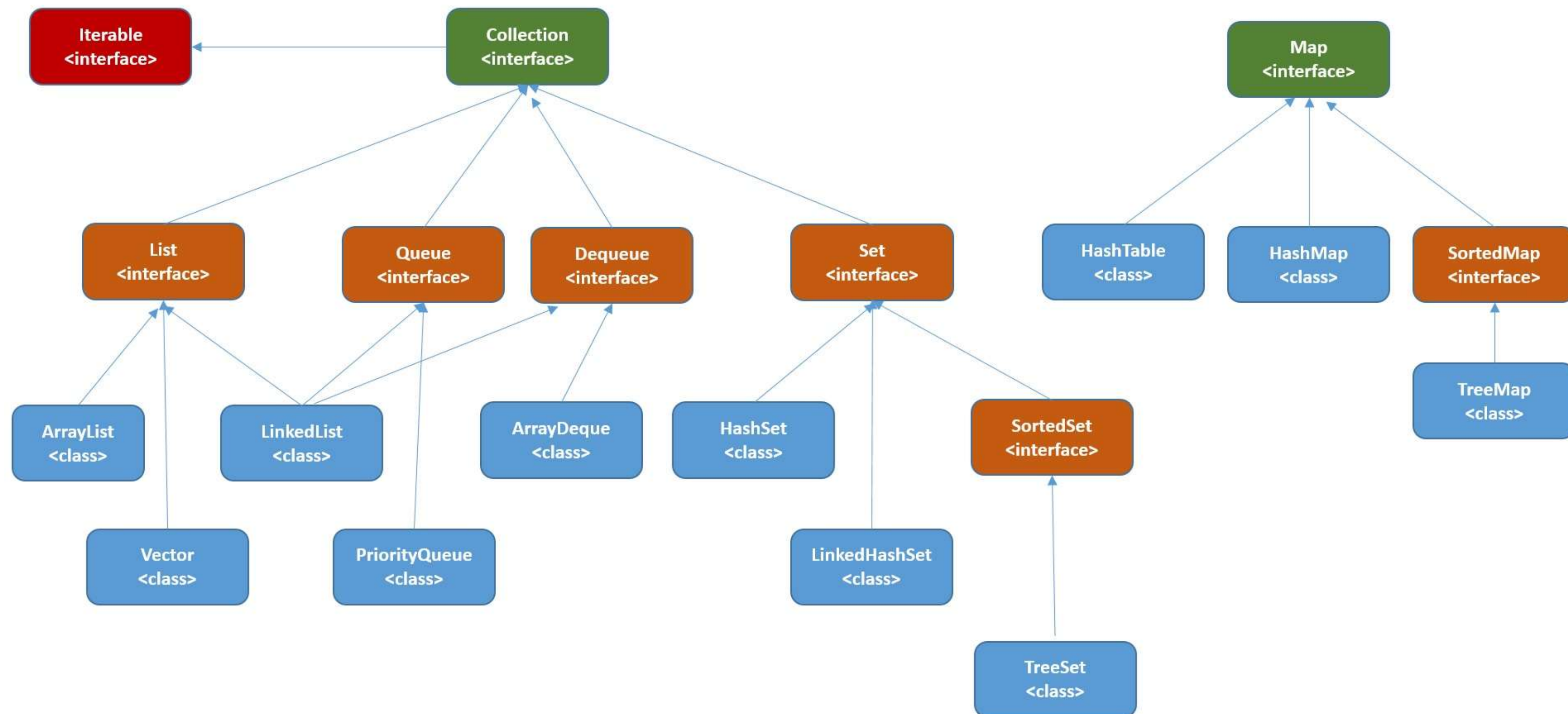
```
List <String> s = List.of("a","b","c","d");
List<String> s = new ArrayList<>(List.of("a","b","c","d"));
```

```
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”, vyhoď 2., a99d

for (ListIterator<String> it = s.listIterator(s.size());
                                it.hasPrevious(); )
   System.out.println(it.previous()); //vypíš zoznam opačne d99a

Set<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]
```

Súbor:ArrayListDemo.java

Immutable Collections

https://docs.oracle.com/javase/9/core/creating-immutable-lists-sets-and-maps.htm#JSCOR-GUID-202D195E-6E18-41F6-90C0-7423B2C9B381

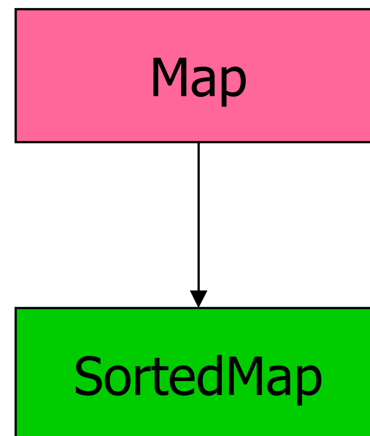


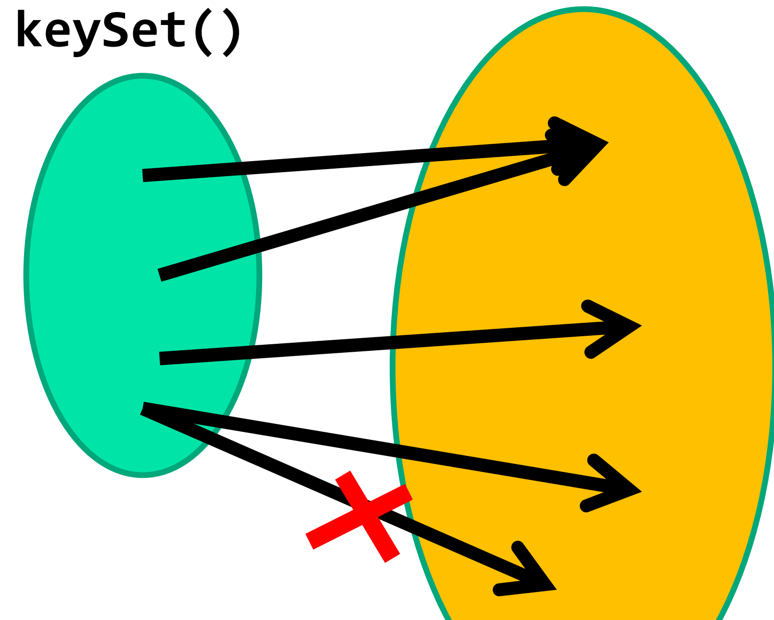
https://facingissuesonit.com/category/collection/

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

zobrazenie : K->V

```
public interface Map<K,V> {
                                        Python: m[key]=value
  V put(K key, V value); // pridaj dvojicu [key;value] do
                         zobrazenia, ak už key->value', tak prepíš
                                        Python: m[key]
  V get(Object key);// nájdi obraz pre key, ak neexistuje->null
  V remove(Object 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;
```

Zobrazenie:String->Integer

```
Map<String, Integer> m = new HashMap<>();
for (String a : args) {  // frekvenčná tabuľka slov v riadku
   Integer freq = m.get(a);  // počet doterajších výskytov
   m.put(a, (freq == null) ? 1 : // ak sa ešte nenachádzal, 1
       freq + 1);                // ak sa nachádzal, +1
}
System.out.println(m);
```

```
java HashMapDemo 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}
```

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

```
Map<String, Integer> m = Map.of(        // ImmutableCollections
                              "one", 1,   // key, value
                              "two", 2,
                              "three", 3
                             );
```

Súbor: HashMapDemo.java

Ako prejsť cez Map

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

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

- ťažší spôsob – iterátorom

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

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

Súbor: HashMapDemo.java

Ak to bude **Tree**Map

(bude to utriedené podľa kľúča)

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

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

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

- Ľahší spôsob

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

- ťažší spôsob

```
for(
var ~~Iterator<AbstractMap.SimpleEntry<String,Integer>>~~ it =
   m.entrySet().iterator();
   it.hasNext(); ) {
     var ~~AbstractMap.SimpleEntry<String,Integer>~~ item = t.next();
       System.out.println("[" + item.getKey() + "]=" +
                                   item.getValue());
} AbstractMap.SimpleEntry<K,V> je trieda implementujúca
Interface Map.Entry<K,V>
```

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

Súbor: HashMapDemo.java

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,
```

Súbor: TreeMapDemo.java

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,...
```

Súbor: TreeMapDemo3.java

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=Pelegrini, Moscow=Putin, Prague=Pavel}
```

Súbor: TreeMapDemo3.java

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,…

Súbor: TreeMapDemo.java

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= Pelegrini, Moscow=Putin, Prague=Pavel}
```

Súbor: TreeMapDemo.java

DFS/BFS/Bactracking

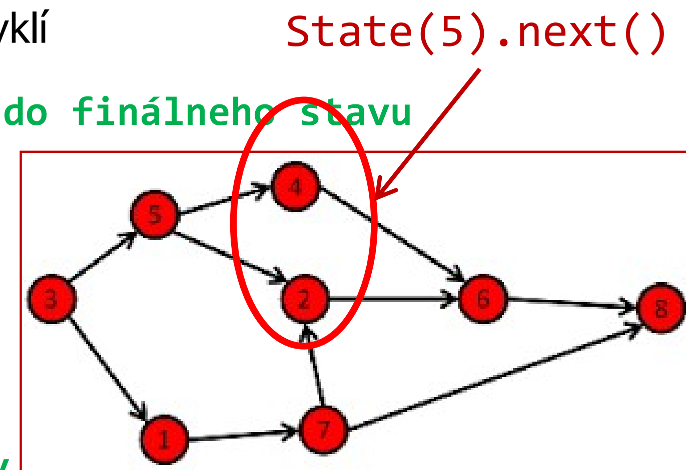
Ide o prehľadávanie stavového priestoru, abstrakcia pre stav môže byť:

```
interface State {
    public State();                          // počiatočný stav hľadania
    abstract boolean isFinalState();         // test na koncový stav hľadania
    abstract State[] next();                 // nasledujúce/susedné stavy
    abstract Set<State> next();              // nasledujúce/susedné stavy
    abstract boolean isCorrect();            // test na korektnosť stavu
}
```

Naivné prehľadávanie pre acyklický graf, ktoré sa na cyklickom zacyklí

State(5).next()

```
public class Search<S extends State> // hľadáme cestu do finálneho stavu
public void searchWhichLoops(S s) {
    if (s.isFinalState())
        add(s); // pridaj do zoznamu riešení
    else
        for (State ns : s.next())
            search(ns);   // rekurzia do všetkých susedov
}
```



Súbor: State, Search.java

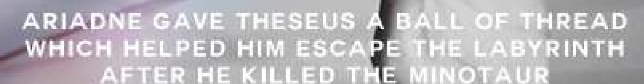
Aby sa to nezacyklilo

(objavila to už Ariadna pri hľadaní Thezea v labyrinte s Minotaurom)

```java
public void search(S s, ArrayList<S> visited) {
    if (s.isFinalState()) {
        add(s);        // pridaj do zoznamu riešení
        add(visited); // pridaj do zoznamu riešení
    } else
        for (State ns : s.next()) {
            if (!visited.contains(ns)) {// nebol si ?
                visited.add(ns); // označ
                search(ns, visited);
                visited.remove(ns); // odznač
            }
        }
}
```

je to depth-first (do hĺbky)
alebo breadth-first (do šírky) ?

Súbor: Search.java



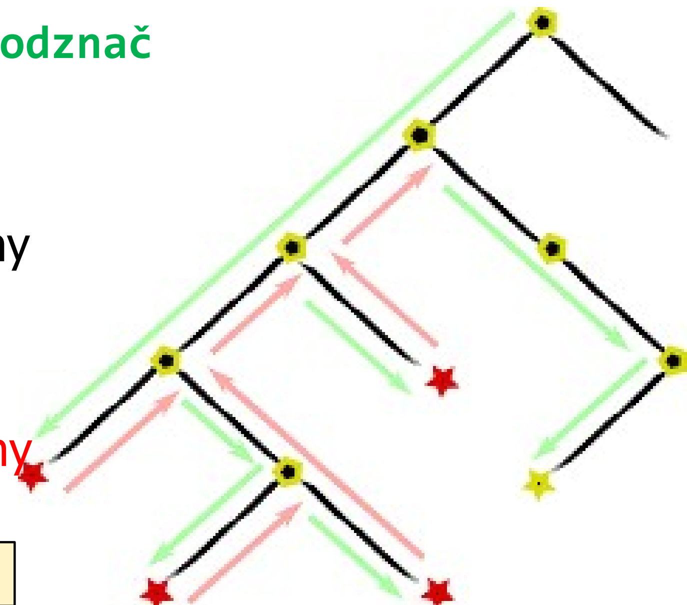
Backtracking

(orezáva podstromy určite neobsahujúce riešenie)

```
public void search(S s, ArrayList<S> visited) {
   if (s.isFinalState())
         add(visited);                    // pridaj do zoznamu riešení
   else
         for (State ns : s.next())  // môže to viesť k riešeniu ?
               if (!visited.contains(ns) && ns.isCorrect()) {
                     visited.add(ns); // označ
                     search((S) ns, visited);
                     visited.remove(ns); // odznač
               }
}
```

Šikovný **`isCorrect`** výrazne zredukuje zväčša exponenciálny prehľadávaný priestor stavov, ale ten aj tak často zostane exponenciálny, ale menší ...

Preto: backtrack nepoužívame na neexponenciálne problémy



Súbor: Search.java

Ako by vyzeral BFS

```
private void search(ArrayList<S> queue, ArrayList<S> visited, boolean DFS) {
    while (queue.size() > 0) {
        S s = queue.remove(0);   // vyber prvý z fronty
        if (s.isFinalState())    // ak si už v cieli
            add(s);              // pridaj do zoznamu riešení
        else
            for (State ns : s.next()) {
                if (!visited.contains(ns) && ns.isCorrect()) {
                    visited.add(ns);
                    if (DFS)                    // ak depth-first search
                        queue.add(0, ns); // pridaj na začiatok fronty
                    else                        // ak breadth-first search
                        queue.add(queue.size(), ns);// pridaj na koniec
                }
            }
    }
}
```

Súbor: Search.java