

# *Seminář Java*

## *VI*

# Rekapitulace

## Kontejnery

- slouží k ukládání objektů (ne hodnot primitivních typů!)
- moderní kontejnery jsou nesynchronizované, nepřípouštějí souběžný přístup z více vláken.
- průchod kontejnery – iterátory
- Seznamy (`ArrayList`)

# Množiny

## Množiny

- struktury standardně bez uspořádání prvků (uspořádané viz dále)
- implementují rozhraní `Set` (což je rozšíření `Collection`)

Cílem množin je mít možnost rychle (se složitostí  $O(\log(n))$ ) provádět atomické operace:

- vkládání prvku (`add`)
- odebírání prvku (`remove`)
- dotaz na přítomnost prvku (`contains`)
- lze testovat i relaci je podmnožinou

## Standardní implementace

- `HashSet` – hašovací tabulka
- `TreeSet` – vyhledávací strom

## Množiny – příklad

Vytvoříme množinu, naplníme ji a testujeme přítomnost prvku

```
public static void main(String[] args) {  
    Set mnozina = new HashSet();  
    Clovek c1 = new Clovek("Ferda");  
    Clovek c2 = new Clovek("Franta");  
  
    mnozina.add(c1);  
    mnozina.add(c2);  
  
    System.out.println("Je v množine Franta?" +  
        mnozina.contains(c2));  
}
```

## Množiny – equals a hashCode

Požadavek: objekty třídy Clovek se stejným obsahem jsou (z hlediska porovnávání) stejné.

```
public static void main(String[] args) {  
    Set mnozina = new HashSet();  
    Clovek c1 = new Clovek("Ferda");  
    Clovek c2 = new Clovek("Franta");  
  
    mnozina.add(c1);  
    mnozina.add(c2);  
  
    Clovek c3 = new Clovek("Franta");  
    System.out.println("Je v mnozine Franta?" +  
        mnozina.contains(c3));  
}
```

# Množiny – equals a hashCode

```
class Clovek {
    String name;
    ...

    public boolean equals(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek) o;
            return name.equals(c.name);
        }
        else
            throw new
                IllegalArgumentException("Nelze porovnat");
    }

    public int hashCode() {
        return name.hashCode();
    }
}
```

# Uspořádané množiny

## Uspořádané množiny

- Implementují rozhraní `SortedSet` (viz *API doc k rozhraní `SortedSet`*)
- Jednotlivé prvky lze tedy iterátorem procházet v přesně definovaném pořadí (uspořádání) podle hodnot prvků.

## Standardní implementace

- `TreeSet` – vyhledávací černobílé stromy (Red-Black Trees)

## Uspořádané množiny – příklad

Vytvoříme množinu, naplníme ji a chodíme po položkách iterátorem.

```
public static void main(String[] args) {
    SortedSet mnozina = new TreeSet();
    Clovek c1 = new Clovek("Ferda");
    Clovek c2 = new Clovek("Franta");

    mnozina.add(c2);
    mnozina.add(c1);

    for (Iterator i = mnozina.iterator(); i.hasNext();)
    {
        Clovek c = (Clovek) i.next();
        c.vypisInfo();
    }
}
```

---

Run-time výjimka `java.lang.ClassCastException`



# Uspořádané množiny – komparátor

Uspořádání je dáno:

- standardním chováním metody `compareTo` vkládaných objektů – pokud implementují rozhraní `Comparable`
- nebo je možné uspořádání definovat pomocí tzv. komparátoru (objektu impl. rozhraní `Comparator`) při vytvoření množiny.

## Uspořádané množiny – komparátor (příklad)

Implementace operace `compareTo`

```
public class Clovek implements Comparable {
    String name;
    ...
    public int compareTo(Object o) {
        if (o instanceof Clovek) {
            Clovek c = (Clovek) o;
            return name.compareTo(c.name);
        }
        else
            throw new
                IllegalArgumentException("Nelze porovnat");
    }
}
```

# Mapy

Mapy (asociativní pole, nepřesně také hašovací tabulky nebo haše) fungují na stejných principech a požadavcích jako Set:

- avšak ukládají dvojice (klíč→hodnota) a umožňují rychlé vyhledání dvojice podle hodnoty klíče,
- základními metodami jsou: dotazy na přítomnost klíče v mapě (`containsKey`),
- výběr hodnoty odpovídající zadanému klíči (`get`),
- možnost získat zvlášť množiny klíčů, hodnot nebo dvojic (klíč→hodnota)

# Mapy

Mapy mají:

- podobné implementace jako množiny (tj. hašovací tabulky nebo stromy)
- logaritmickou složitost základních operací
  - `put`
  - `remove`
  - `containsKey`

Standardní implementace

- `HashSet`
- `TreeMap`

## Mapy – příklad

Vytvoříme množinu, naplníme ji a chodíme po položkách iterátorem.

```
public static void main(String[] args) {  
    Map mapa = new HashMap();  
    Clovek c1 = new Clovek("Ferda");  
    Clovek c2 = new Clovek("Franta");  
  
    mapa.put(c1.name, c1);  
    mapa.put(c2.name, c2);  
  
    Clovek c = (Clovek) mapa.get("Ferda");  
    c.vypisInfo();  
}
```

# Uspořádané mapy

## Uspořádané mapy

- Implementují rozhraní `SortedMap`
- Dvojice (klíč→hodnota) jsou v nich uspořádané podle hodnot klíče.
- Uspořádání lze ovlivnit naprosto stejným postupem jako u uspořádané množiny.

## Standardní implementace

- `TreeMap` – implementace Red-Black Trees

## Uspořádané mapy – příklad

Vytvoříme množinu, naplníme ji a chodíme po položkách iterátorem.

```
public static void main(String[] args) {
    SortedMap mapa = new TreeMap();
    Clovek c1 = new Clovek("Ferda");
    Clovek c2 = new Clovek("Franta");
    Clovek c3 = new Clovek("Cecil");

    mapa.put(c1.name, c1);
    mapa.put(c2.name, c2);
    mapa.put(c3.name, c3);

    for (Iterator i = mapa.keySet().iterator(); i.hasNext(); )
    {
        String jmeno = (String) i.next();
        Clovek c = (Clovek) mapa.get(jmeno);
        c.vypisInfo();
    }
}
```

## Uspořádané mapy – příklad

```
Exception in thread "main" java.lang.ClassCastException: ...  
    at java.util.TreeMap.compare(TreeMap.java:1093)  
    at java.util.TreeMap.put(TreeMap.java:465)  
    at seminar6.SortedMapDemo.main(SortedMapDemo.java:14)
```



## Uspořádané mapy – příklad s komparátorem

Vytvoříme množinu, naplníme ji a chodíme po položkách iterátorem.

```
public static void main(String[] args) {  
    SortedMap mapa = new TreeMap(new ClComparator());  
  
    Clovek c1 = new Clovek("Ferda");  
    Ucet u1 = new Ucet(500);  
  
    Clovek c2 = new Clovek("Franta");  
    Ucet u2 = new Ucet(100);  
  
    Clovek c3 = new Clovek("Cecil");  
    Ucet u3 = new Ucet(300);  
  
    mapa.put(c1, u1);  
    mapa.put(c2, u2);  
    mapa.put(c3, u3);  
    ...  
}
```

## Uspořádané mapy – příklad s komparátorem

```
...  
for (Iterator i = mapa.keySet().iterator(); i.hasNext();)  
{  
    Clovek majitel = (Clovek) i.next();  
    Ucet ucet = (Ucet) mapa.get(majitel);  
    majitel.vypisInfo();  
    System.out.println(" je majitelem uctu se " +  
        "zustatkem " + ucet.zustatek + " Kc");  
}  
...
```

## Uspořádané mapy – příklad s komparátorem

```
...
static class Ucet {
    double zustatek;
    public Ucet(double z) {
        zustatek = z;
    }
}

static class Clovek { // nemusí být Comparable
    String name;
    Clovek (String n) {
        name = n;
    }
    public void vypisInfo() {
        System.out.print("Clovek "+name);
    }
}
...
```

## Uspořádané mapy – příklad s komparátorem

```
...
static class ClComparator implements Comparator
{
    public int compare(Object o1, Object o2)
    {
        // porovnává jen lidi podle jmena
        if (o1 instanceof Clovek && o2 instanceof Clovek)
        {
            Clovek c1 = (Clovek)o1;
            Clovek c2 = (Clovek)o2;
            return c1.name.compareTo(c2.name);
        } else
            throw new IllegalArgumentException(
                "Nelze porovnat ...");
    }
}
```

# Srovnání implementací kontejnerů

## Seznamy

- na bázi pole (`ArrayList`) – rychlý přímý přístup (přes index)
- na bázi lineárního zřetězeného seznamu (`LinkedList`) – rychlý sekvenční přístup (přes iterátor)
- téměř vždy se používá `ArrayList` – stejně rychlý a paměťově efektivnější

## Množiny a mapy

- na bázi hašovacích tabulek (`HashMap`, `HashSet`) – rychlejší, ale neuspořádané (lze získat iterátor procházející klíče uspořádaně)
- na bázi vyhledávacích stromů (`TreeMap`, `TreeSet`) – pomalejší, ale uspořádané
- spojení výhod obou – `LinkedHashSet`, `LinkedHashMap` (novinka v Javě 2, v1.4)

# Historie

Existují tyto starší typy kontejnerů ( $\Rightarrow$  náhrada)

- `Hashtable`  $\Rightarrow$  `HashMap`, `HashSet` (podle účelu)
- `Vector`  $\Rightarrow$  `List`
- `Stack`  $\Rightarrow$  `List`

Roli iterátoru plnil dříve výčet (enumeration) se dvěma metodami:

- `boolean hasMoreElements()`
- `Object nextElement()`

# Generics (Java 5.0)

## Problém při práci s kontejnery

- při vybírání elementu z kolekce musíme *přetypovat*
- není hlídáno kompilátorem
- pouze dynamická typová kontrola
- ⇒ šance na vygenerování run-time výjimky

## Generics (vlastnost Java 5.0)

- umožňuje komunikaci s kompilátorem
- statická kontrola typů při manipulaci s kontejnery
- bez přetypování!

## Generics (Java 5.0)

```
// Removes 4-letter words from c. Elements must be strings.
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
        if (((String) i.next()).length() == 4)
            i.remove();
}
```

---

```
// Removes the 4-letter words from c
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator(); i.hasNext(); )
        if (i.next().length() == 4)
            i.remove();
}
```



## The For-Each Loop (Java 5.0)

```
void cancelAll(Collection<TimerTask> c) {  
    for (Iterator<TimerTask> i = c.iterator(); i.hasNext();)  
        i.next().cancel();  
}
```

---

```
void cancelAll(Collection<TimerTask> c) {  
    for (TimerTask t : c)  
        t.cancel();  
}
```

## The For-Each Loop (Java 5.0)

```
List suits = ...;
List ranks = ...;
List sortedDeck = new ArrayList();

// BROKEN - throws NoSuchElementException!
for (Iterator i = suits.iterator(); i.hasNext(); )
    for (Iterator j = ranks.iterator(); j.hasNext(); )
        sortedDeck.add(new Card(i.next(), j.next()));



---



// Fixed, though a bit ugly
for (Iterator i = suits.iterator(); i.hasNext(); ) {
    Suit suit = (Suit) i.next();
    for (Iterator j = ranks.iterator(); j.hasNext(); )
        sortedDeck.add(new Card(suit, j.next()));
}
```

## The For-Each Loop (Java 5.0)

```
List suits = ...;
List ranks = ...;
List sortedDeck = new ArrayList();

for (Suit suit : suits)
    for (Rank rank : ranks)
        sortedDeck.add(new Card(suit, rank));
```

## Generics a For-Each Loop (Java 5.0)

```
for (Iterator i = mapa.keySet().iterator(); i.hasNext();)  
{  
    Clovek majitel = (Clovek) i.next();  
    Ucet ucet = (Ucet) mapa.get(majitel);  
    majitel.vypisInfo();  
    System.out.println(" je majitelem uctu se " +  
        "zustatkem " + ucet.zustatek + " Kc");  
}
```

---

`SortedMap<Clovek,Ucet> = new TreeMap<Clovek,Ucet>...`

```
for(Clovek majitel : mapa.keySet())  
{  
    Ucet ucet = mapa.get(majitel);  
    majitel.vypisInfo();  
    System.out.println(" je majitelem uctu se " +  
        "zustatkem " + ucet.zustatek + " Kc");  
}
```

# The For-Each Loop (Java 5.0)

Dá se použít i pro pole ...

```
// Returns the sum of the elements of a
int sum(int[] a) {
    int result = 0;
    for (int i : a)
        result += i;
    return result;
}
```

## Autoboxing (Java 5.0)

```
// Prints a frequency table of the words on the command line
public class Frequency1 {
    public static void main(String[] args) {
        Map m = new TreeMap();
        for (int i=0; i<args.length; i++) {
            String word = args[i];
            Integer freq = (Integer) m.get(word);
            m.put(word, (freq == null ? new Integer(1) :
                new Integer(freq.intValue() + 1)));
        }
        System.out.println(m);
    }
}
```

---

```
java Frequency1 if it is to be it is up to me to do
{be=1, do=1, if=1, is=2, it=2, me=1, to=3, up=1}
```

## Autoboxing (Java 5.0)

```
// Prints a frequency table of the words on the command line
public class Frequency {
    public static void main(String[] args) {
        Map<String, Integer> m = new
                                TreeMap<String, Integer>();
        for (String word : args) {
            Integer freq = m.get(word);
            m.put(word, (freq == null ? 1 : freq + 1));
        }
        System.out.println(m);
    }
}
```

---

```
java Frequency1 if it is to be it is up to me to do
{be=1, do=1, if=1, is=2, it=2, me=1, to=3, up=1}
```

# Odkazy

- Podrobné seznámení s kontejnery  
<http://java.sun.com/docs/books/tutorial/collections/>
- Popis Java 5.0  
<http://java.sun.com/j2se/1.5.0/docs/index.html>