Kapitola 1. Přednáška 8 - dynamické datové struktury (kontejnery). Generické typy.

Obsah

Kontejnery

- Kontejnery jako základní dynamické struktury v Javě
- Kolekce, iterátory (Collection, Iterator)
- Seznamy (rozhraní List, třídy ArrayList, LinkedList)
- Množiny (rozhraní Set, třída HashSet), uspořádané množiny (rozhraní SortedSet, třída TreeSet), roz-

hraní Comparable, Comparator

- Mapy (rozhraní Map, třída HashMap), uspořádané mapy (rozhraní SortedMap, třída TreeMap)
- Klasické netypové vs. nové typové kontejnery - generické datové typy
- Iterace cyklem foreach
- Starší typy kontejnerů (Vector, Stack, Hashtable)

Kontejnery

Kontejnery (containers) v Javě

- slouží k ukládání objektů (ne hodnot primitivních typů!)
- v Javě koncipovány dosud jako *beztypové* - to se ve verzi 1.5 částečně změní!
- tím se liší od např. *Standard Template Library* v C++

Většinou se používají kontejnery hotové, vestavěné, tj. ty, jež jsou součastí Java Core API:

- vestavěné kontejnerové třídy jsou definovány v balíku `java.util`
- je možné vytvořit si vlastní implementace, obvykle ale zachovávající/implementující „standardní“ rozhraní

K čemu slouží?

- jsou dynamickými alternativami k poli a mají daleko širší použití
- k uchování proměnného počtu objektů -
- počet prvků se v průběhu existence kontejneru může měnit
- oproti polím nabízejí časově efektivnější algoritmy přístupu k prvkům

Základní kategorie kontejnerů

- seznam (`List`) - lineární struktura
- množina (`Set`) - struktura bez uspořádání, rychlé dotazování na přítomnost prvku
- asociativní pole, mapa (`Map`) - struktura uchovávající dvojice klíč->hodnota, rychlý přístup přes klíč

Kontejnery - rozhraní, nepovinné metody

- Funkcionalita vestavěných kontejnerů je obvykle předepsána výhradně *rozhraním*, jenž implementují.
- Rozhraní však připouštějí, že některé metody jsou *nepovinné*, třídy jej nemusí implementovat!
- V praxi se totiž někdy nehodí implementovat jak čtecí, tak i zápisové operace - některé kontejnery jsou „read-only“

Kontejnery - souběžný přístup, výjimky

- Moderní kontejnery jsou *nesynchronizované*, nepřipouštějí souběžný přístup z více vláken.
- Standardní, nesynchronizovaný, kontejner lze však „zabalit“ synchronizovanou obálkou.
- Při práci s kontejnery může vzniknout řada *výjimek*, např. `IllegalStateException` apod.
- Většina má charakter výjimek *běhových*, není povinností je odchytávat - pokud věříme, že nevzniknout.

Iterátory

Iterátory jsou prostředkem, jak "chodit" po prvcích kolekce buďto

- v neurčeném pořadí nebo
- v uspořádání (u uspořádaných kolekcí)

Každý iterátor musí implementovat velmi jednoduché rozhraní `Iterator` se třemi metodami:

- `boolean hasNext()`
- `Object next()`
- `void remove()`

Kolekce

Kolekce

- jsou kontejnery implementující rozhraní `Collection` - API doc k rozhr. `Collection`

[http://java.sun.com/j2se/1.4/docs/api/java/util/Collection.html]

- Rozhraní kolekce popisuje velmi obecný kontejner, disponující operacemi: *přidávání, rušení prvku, získání iterátoru, zjišťování prázdnosti* atd.
- Mezi kolekce patří mimo `Map`y všechny ostatní vestavěné kontejnery - `List`, `Set`
- Prvky kolekce nemusí mít svou pozici danou indexem - viz např. `Set`

Seznamy

- lineární struktury
- implementují rozhraní `List` (rozšíření `Collection`) API doc k rozhr. `List` [http://java.sun.com/j2se/1.4/docs/api/java/util/List.html]
- prvky lze adresovat indexem (typu int)
- poskytují možnost získat dopředný i zpětný *iterátor*
- lze pracovat i s *podseznamy*

Seznamy a obyčejné iterátory - příklad

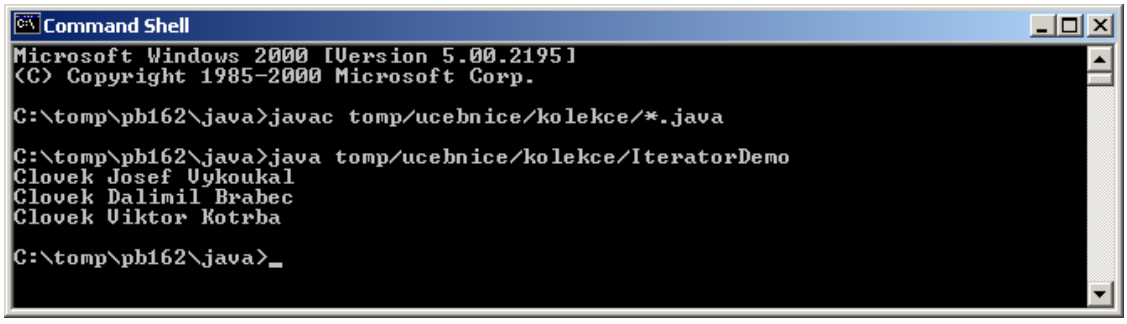
Vytvoříme seznam, naplníme jej a chodíme po položkách iterátorem.

Obrázek 1.1. Pohyb po seznamu iterátorem

IteratorDemo.java

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme prázdný seznam
        List seznam = new ArrayList();

        // naplníme jej třemi lidmi
        seznam.add(new Clovek("Josef", "Vykoukal"));
        seznam.add(new Clovek("Dalimil", "Brabec"));
        seznam.add(new Clovek("Viktor", "Kotrba"));

        // projdeme iterátorem a vypíšeme postupně všechny lidi
        for(Iterator i = seznam.iterator(); i.hasNext(); ) {
            // získej další objekt: je to člověk, můžeš přetypovat
            Clovek c = (Clovek)i.next();
            c.vypisInfo();
        }

    }
    // Clovek bude vnořená třída, aby nebyla vůbec vidět ven -
    // - jiný význam to nemá...
    static class Clovek {
        String jmeno, prijmeni;
        Clovek (String j, String p) {
            jmeno = j;
            prijmeni = p;
        }
        public void vypisInfo() {
            System.out.println("Clovek "+jmeno+" "+prijmeni);
        }
    }
}
```

Obrázek 1.2. Spuštění pg. s iterátorem



Iterátor po seznamu - příklad

Vytvoříme seznam, naplníme jej a chodíme po položkách seznamovým iterátorem, vytvořeným na určité pozici (indexu) v seznamu.

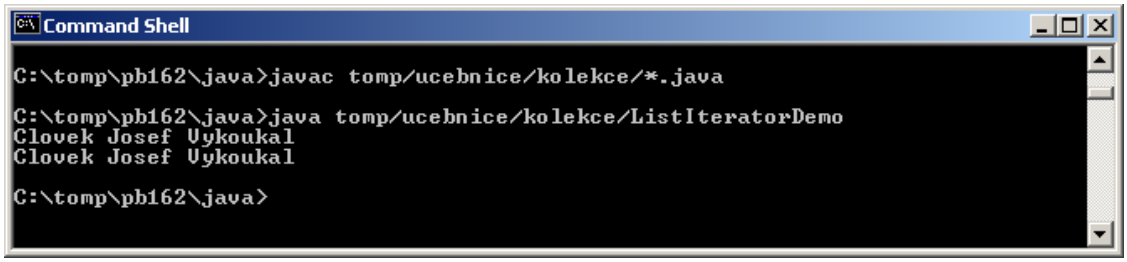
Obrázek 1.3. Pohyb seznamovým iterátorem

ListIteratorDemo.java

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

    // vytvoříme prázdný seznam
    List seznam = new ArrayList();

    // naplníme jej třemi lidmi
    seznam.add(new Clovek("Josef", "Vykoukal"));
    seznam.add(new Clovek("Dalimil", "Brabec"));
    seznam.add(new Clovek("Viktor", "Kotrba"));

    // získáme seznamový iterátor od pozice 1
    // - umí víc než obyčejný!
    ListIterator li = seznam.listIterator(1);

    // posuň se pozpátku na pozici 0
    // a získej objekt z pozice 0
    Clovek c = (Clovek)li.previous();
    c.vypisInfo();

    // získej aktuální objekt z pozice 0
    // a posuň se na 1
    c = (Clovek)li.next();
    c.vypisInfo();

}
```

K procházení seznamovým iterátorem lze použít metody `next`, `previous`.

Obrázek 1.4. Spuštění pg. se seznamovým iterátorem



Množiny

Množiny

- jsou struktury standardně bez uspořádání prvků (ale existují i 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 množiny:

- *hašovací tabulka* (`HashSet`) nebo
- *vyhledávací strom* (černobílý strom, Red-Black Tree - `TreeSet`)

Množina - příklad

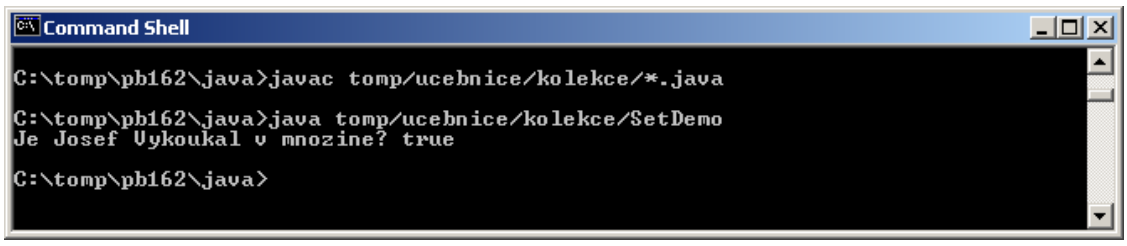
Vložíme prvky do množiny a ptáme se, zda tam jsou:

Obrázek 1.5. Vložení prvků do množiny a dotaz na přítomnost

SetDemo.java

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme prázdnou množinu
        Set mnozina = new HashSet();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        mnozina.add(c1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        mnozina.add(c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        mnozina.add(c3);

        System.out.println("Je Josef Vykoukal v mnozine? "
                           +mnozina.contains(c1));
```

Obrázek 1.6. Spuštění pg. s množinou



Uspořádané množiny

Uspořádané množiny:

- Implementují rozhraní `SortedSet` -API doc k rozhraní `SortedSet` [http://java.sun.com/j2se/1.4/docs/api/java/util/SortedSet.html]
- Jednotlivé prvky lze tedy iterátorem procházet v přesně definovaném pořadí - uspořádání podle *hodnot prvků*.

- Existuje vestavěná impl. `TreeSet` - černobílé stromy (Red-Black Trees) API doc ke třídě `TreeSet` [http://java.sun.com/j2se/1.4/docs/api/java/util/TreeSet.html]

Uspořádání je dáno buďto:

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

Uspořádaná množina - příklad s chybou

Vložíme prvky do uspořádané množiny. Prvky musejí implementovat rozhraní Comparable, nebo musíme poskytnout komparátor. Když neuděláme ani jedno:

Obrázek 1.7. Vložení neporovnatelných prvků do uspořádané množiny

SortedSetDemo.java | ListIteratorDemo

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme uspořádanou množinu
        SortedSet um = new TreeSet();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        um.add(c1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        um.add(c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        um.add(c3);

        for(Iterator i = um.iterator(); i.hasNext(); ) {
            // získej další objekt: je to člověk, můžeš př
            Clovek c = (Clovek)i.next();
            c.vypisInfo();
        }
    }
```

Obrázek 1.8. Spuštění nefunkčního pg. s uspořádanou množinou

Command Shell

```
C:\tomp\pb162\java>java tomp/ucebnice/kolekce/SortedSetDemo
Exception in thread "main" java.lang.ClassCastException
        at java.util.TreeMap.compare(TreeMap.java:1085)
        at java.util.TreeMap.put(TreeMap.java:463)
        at java.util.TreeSet.add(TreeSet.java:209)
        at tomp.ucebnice.kolekce.SortedSetDemo.main(SortedSetDemo.java:16)

C:\tomp\pb162\java>
```

Nefunguje, prvky třídy `Clovek` nebyly porovnatelné.

Uspořádaná množina - příklad OK

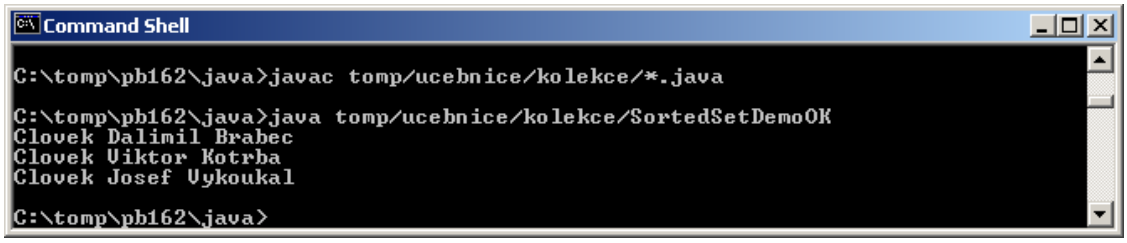
Prvky implementují rozhraní Comparable:

Obrázek 1.9. Vložení porovnatelných prvků do uspořádané množiny

SortedSetDemoOK.java | ListIteratorDemo

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme uspořádanou množinu
        SortedSet um = new TreeSet();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        um.add(c1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        um.add(c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        um.add(c3);

        for(Iterator i = um.iterator(); i.hasNext(); ) {
            // získej další objekt: je to člověk, můžeš přetypovat
            Clovek c = (Clovek)i.next();
            c.vypisInfo();
        }
    }
    static class Clovek implements Comparable {
        String jmeno, prijmeni;
        Clovek (String j, String p) {
            jmeno = j;
            prijmeni = p;
        }
        public void vypisInfo() {
            System.out.println("Clovek "+jmeno+" "+prijmeni);
        }
        public int compareTo(Object o) {
            if (o instanceof Clovek) {
                Clovek c = (Clovek)o;
                return prijmeni.compareTo(c.prijmeni);
            } else
                throw new IllegalArgumentException(
                "Nelze porovnat objekt typu Clovek s objektem jineho typu")
        }
    }
}
```

Obrázek 1.10. Spuštění funkčního pg. s uspořádanou množinou



Funguje, prvky třídy `Clovek` jsou porovnatelné, množina je uspořádána podle příjmení lidí.

Mapy

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

- Ukládají ovšem dvojice (klíč, hodnota) a umožnují 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 mají:

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

Mapa - příklad

Lidi se do mapy vkládají s klíčem = příjmení člověka, pak se přes příjmení mohou vyhledat:

Obrázek 1.11. Vložení lidí do mapy pod klíčem příjmení a vyhledání člověka

MapDemo.java | ListIteratorDemo

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme mapu
        Map m = new HashMap();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        m.put(c1.prijmeni, c1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        m.put(c2.prijmeni, c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        m.put(c3.prijmeni, c3);

        // ziskej Kotrbu
        Clovek c = (Clovek)m.get("Kotrba");
        c.vypisInfo();
    }
    static class Clovek {
        String jmeno, prijmeni;
        Clovek (String j, String p) {
            jmeno = j;
            prijmeni = p;
        }
        public void vypisInfo() {
            System.out.println("Clovek "+jmeno+" "+prijmeni);
        }
    }
}
```

Obrázek 1.12. Spuštění funkčního pg. s mapou

```
Command Shell

C:\tomp\pb162\java>javac tomp/ucebnice/kolekce/*.java

C:\tomp\pb162\java>java tomp/ucebnice/kolekce/MapDemo
Clovek Viktor Kotrba

C:\tomp\pb162\java>
```

Uspořádané mapy

Uspořádané mapy:

- Implementují rozhraní `SortedMap` – API doc k rozhraní `SortedMap` [http://java.sun.com/j2se/1.4/docs/api/java/util/SortedMap.html]
- Dvojice (klíč, hodnota) jsou v nich *uspořádané podle hodnot klíče*.
- Existuje vestavěná impl. `TreeMap` - černobílé stromy (Red-Black Trees) - API doc ke třídě `TreeMap` [http://java.sun.com/j2se/1.4/docs/api/java/util/TreeMap.html]
- Uspořádání lze ovlivnit naprosto stejným postupem jako u uspořádané množiny.

Uspořádaná mapa - příklad

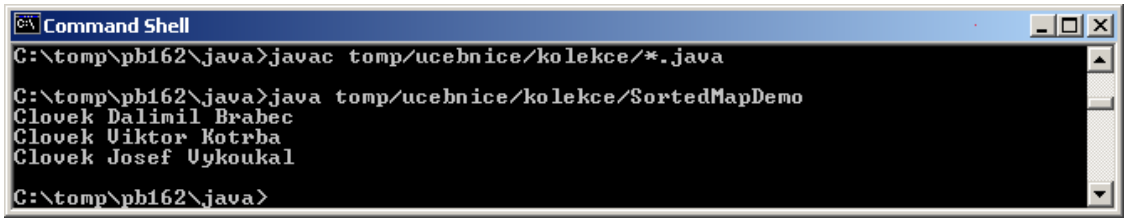
Jsou-li klíče uspořádané (pomocí implementace Comparable nebo komparátorem), mohou se prvky procházet podle uspořádání klíčů:

Obrázek 1.13. Vložení lidí do mapy pod uspořádaným klíčem příjmení a projde je

SortedMapDemo.java | ListIteratorDemo

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme mapu
        SortedMap sm = new TreeMap();

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        sm.put(c1.prijmeni, c1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        sm.put(c2.prijmeni, c2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        sm.put(c3.prijmeni, c3);

        // projdi abecedně všechny lidi v mapě
        // proto je třeba získat iterátor nad množinou klíčů
        for(Iterator i = sm.keySet().iterator(); i.hasNext(); ) {
            String prij = (String)i.next();
            Clovek c = (Clovek)sm.get(prij);
            c.vypisInfo();
        }
    }
    static class Clovek {
        String jmeno, prijmeni;
        Clovek (String j, String p) {
            jmeno = j;
            prijmeni = p;
        }
        public void vypisInfo() {
            System.out.println("Clovek "+jmeno+" "+prijmeni);
        }
    }
}
```

Obrázek 1.14. Spuštění funkčního pg. s uspořádanou mapou



Uspořádaná mapa - příklad s komparátorem

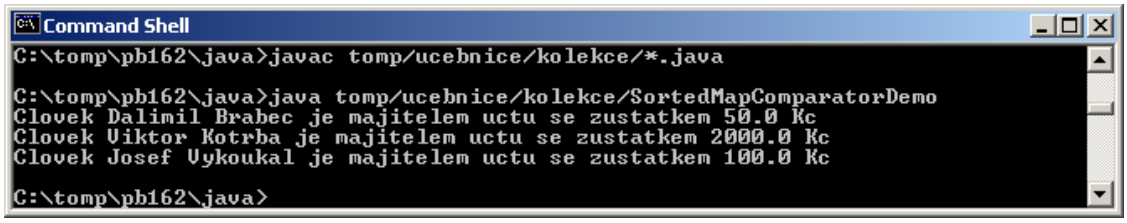
Příklad, kdy jsou klíče uspořádané komparátorem:

Obrázek 1.15. Vložení účtů do mapy pod uspořádaným klíčem člověka - vlastníka

```
package tomp.ucebnice.kolekce;

import java.util.*;

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

        // vytvoříme mapu
        SortedMap sm = new TreeMap(new ClovekComparator());

        // naplníme ji třemi lidmi
        Clovek c1 = new Clovek("Josef", "Vykoukal");
        Ucet   u1 = new Ucet(100);
        sm.put(c1, u1);

        Clovek c2 = new Clovek("Dalimil", "Brabec");
        Ucet   u2 = new Ucet(50);
        sm.put(c2, u2);

        Clovek c3 = new Clovek("Viktor", "Kotrba");
        Ucet   u3 = new Ucet(2000);
        sm.put(c3, u3);

        // projdi abecedně všechny vlastníky účtů v mapě
        // proto je třeba získat iterátor nad množinou klíčů
        for(Iterator i = sm.keySet().iterator(); i.hasNext(); ) {
            Clovek majitel = (Clovek)i.next();
            Ucet ucet = (Ucet)sm.get(majitel);
            majitel.vypisInfo();
            System.out.println(" je majitelem uctu se zustatkem "
                               + ucet.zustatek + " Kc");
        }
    }
    static class Ucet {
        double zustatek;
        public Ucet(double z) {
            zustatek = z;
        }
    }
```

```
    static class Clovek { // nemusí být Comparable
        String jmeno, prijmeni;
        Clovek (String j, String p) {
            jmeno = j;
            prijmeni = p;
        }
        public void vypisInfo() {
            System.out.print("Clovek "+jmeno+" "+prijmeni);
        }
    }
    static class ClovekComparator implements Comparator {
        public int compare(Object o1, Object o2) {
            // porovnává jen lidi a to podle příjmení
            if (o1 instanceof Clovek && o2 instanceof Clovek) {
                Clovek c1 = (Clovek)o1;
                Clovek c2 = (Clovek)o2;
                return c1.prijmeni.compareTo(c2.prijmeni);
            } else
                throw new IllegalArgumentException(
                "Nelze porovnat objekt typu Clovek s objektem jineho typu");
        }
    }
}
```

Obrázek 1.16. Spuštění funkčního pg. s uspořádanou mapou



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:října

- 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 (-> náhrada):

- `Hashtable` -> `HashMap`, `HashSet` (podle účelu)
- `Vector` -> `List`
- `Stack` -> `List`

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

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

Odkazy

Demo efektivity práce kontejnerů - Demo kolekcí [http://www.fi.muni.cz/~tomp/java/ucebnice/javasrc/tomp/ucebnice/kolekce/Kolekce.java]

Velmi podrobné a kvalitní seznámení s kontejnery najdete na Trail: Collections [http://java.sun.com/docs/books/tutorial/collections/index.html]

Zadání úlohy 6.

...zde bude uvedeno zadání 6. úlohy...

Pozn: Pokud cvičící zadání modifikuje, je to OK. Tohle je *vzorové zadání*. Za úlohu získáte opět max. **5 bodů**.