Seminář Java

V

Rekapitulace

- Hierarchie dědičnosti
- Operátory a výrazy
- Porovnání objektu
  - `hashCode()`, `equals()`
- Vyjímky
- Ladění programu

Obsah

- Úvod do kontejnerů - kategorie
- Iterátory
- Rozhraní List
- Rozhraní Set
- Rozhraní Map
- Volba implementace
- Nástroje
- Souběžný přístup

Co jsou to kontejnery

Kontejner je objekt schopný uchovávat skupinu elementů (jiných objektů). Tento objekt pak poskytuje metody pro operace nad těmito daty.

Obsahuje řešení pro vestavěné datové struktury jako jsou:

- Tříděná pole
- Kolekce
- Hašované tabulky
- Různé typy seznamů
- ...

Tyto datové struktury jsou k dispozici v základním balíku **java.util**.

Kategorie

- Kolekce (List)
  - určitá společná pravidla
  - seznamy udržují prvky v pořadí
  - množiny (Set) nesmí obsahovat žádné duplicitní položky
- Mapa (někdy též nazývano jako asociativní pole)
  - skupina dvojic objektů klíč-hodnota

Collection framework

Základ celého systému kolekcí je tzv. `Collection framework`. Je to soubor veřejných rozhraní a dalších typů tříd, které můžeme použít při implementaci kolekcí vlastních.

- Collection - skupina elementů
- Set - skupina elementů bez duplicit
- SortedSet - Set, ale elementy jsou seřazeny
- List - seřazená kolekce nebo seznam
- Map - objekt mapující dvojice objektů klíč-hodnota
- SortedMap - Map, avšak prvky jsou seřazeny
- ArrayList - List jako pole prvků
- LinkedList - List sloužící jako linkovaný seznam

Collection framework

- HashSet - Set jako hašovací tabulka
- TreeSet - SortedSet používaný jako strom
- HashMap - Map s klíči v hašovací tabulce
- TreeMap - SortedMap používaný jako strom s prvky a klíči

Příklad 1 - naplnění a tisk kontejneru

```
static Collection fill(Collection c) {
   c.add("Apple");
   c.add("Apple");
   c.add("Pear");
   return c;
}

static Map fill(Map m) {
   m.put("Apple", "Green");
   m.put("Apple", "Red");
   m.put("Pear", "Yellow");
   return m;
}

public static void main(String[] args) {
    System.out.println(fill(new ArrayList()));
    System.out.println(fill(new HashSet()));
    System.out.println(fill(new HashMap()));
}
```

Příklad 1 - výstup

```
[Apple, Apple, Pear]
[Apple, Pear]
{Apple=Red, Pear=Yellow}
```

Příklad 1 - upřesnění List, Set

Implicitní chování při tisku je zajišťování metodou `toString()` každého kontejneru.

List

- uchovává objekty přesně tak jak byly vloženy
- nepřeskupuje je, ani jen neupravuje

Set

- od každého prvku pouze jednu položku
- používá vlastní interní metodu řazení
- zajímá nás existence, ne pořadí

Položky do libovolné kolekce přidávájí pomocí `add()`.

Příklad 1 - upřesnění Map

Map

- od každého prvku pouze jednu položku
- založený na klíčí
- vlástní metoda třídění

Do mapy přidáváme položky metodou `put()`.

Doporučení

Při vytváření typu kolekce, neukládejte odkaz do proměnné typu této třídy, ale do proměnné typu jejího základního rozhraní.

```
TreeSet ts = new TreeSet();     //stejný typ
SortedSet ts = new TreeSet();   //odkaz na rozhraní
```

Tímto způsobem jsme připraveni na možnou záměnu třídy TreeSet na jinou s rozhraním SortedSet, aniž by byl zbytek kódu touto změnou ovlivněn.

Charakter kontejnerů

- Objekt po vložení do kontejneru ztrácí typovou informaci
  - neexistuje žádné omezení týkající se typu objektů, které lze do kontejneru ukládat
- Neexistuje typová kontrola
  - chceme-li objekt v uložený v kontejneru použít, musíme jej nejprve přetypovat
- Počet prvků v kontejneru se během jeho existence může měnit
- Parametrizované typy od verze `JDK 1.5+`

Příklad 2 - neznámý typ

```
public class Apple {
  private String color;
  Apple(String c) {
    color = c;
  }
  void print() {
    System.out.println(color + " Apple");
  }
}

public class Pear {
  private String color;
  Pear(String c) {
    color = c;
  }
  void print() {
    System.out.println(color + " Pear");
  }
}
```

Příklad 2 - neznámý typ

```
public class ApplesAndPears {

  public static void main(String[] args) {
    ArrayList fruit = new ArrayList();
    fruit.add(new Apple("Green"));
    fruit.add(new Apple("Red"));
    fruit.add(new Pear("Yellow"));//k jablkům přidáme hrušku

    for(int i = 0; i < fruit.size(); i++)
      ((Apple) fruit.get(i)).print();//ClassCastException
  }
}
```

Iterátory

Otázka

Můžeme používat kontejner aniž bysme znali jeho přesný typ? Například začít s kontejnerem typu `ArrayList` a později jej změnit na zřetězený seznam.

- Obecné řešení, které se nezajímá o typ kontejneru s nímž bude pracovat
- Použít iterátory

Iterátor - objekt, jehož úkolem je procházet souvislou řadu objektů a vybírat z této řady objekty, aniž by musel programátor znát vnitřní strukturu této řady.

Operace objektu `Iterator`

- požádat kontejner pomocí metody `iterator()` o předání objektu `Iterator`. Tento iterátor vrátí první prvek množiny objektů, uložených v kontejneru, jakmile poprvé zavoláme metodu `next()`
- získat další objekt této množiny pomocí dalšího volání metody `next()`
- pomocí metody `hasNext()` zjistit, zda množina uložených objektů obsahuje ještě další objekty
- pomocí metody `remove()` vyjmout poslední prvek vrácený iterátorem
- některé implementace iterátoru jsou schopny pohybu pouze v jednom směru

Příklad 3 - neznámý typ, pomocí iterátoru

```
public class ApplesAndPears {
  public static void main(String[] args) {
    ArrayList fruit = new ArrayList();
    fruit.add(new Apple("Green"));
    fruit.add(new Apple("Red"));
    fruit.add(new Pear("Yellow"));//k jablkům přidáme hrušku

    while(fruit.hasNext())
      ((Apple) fruit.next()).print();//ClassCastException
  }
}
```

Rozhraní List

List (rozhraní)

- Pořadí je nejdůležitější vlastností seznamu
- Rozšiřuje rozhraní Collection o celou řadu dalších metod

Obsahuje dva typy seznamů

- základní `ArrayList` jenž vyniká při přímém přístupu k jednotlivým prvkům
  - ukládání a odstraňování prvků je pomalé a neefektivní
- výkonnější `LinkedList` zřetězený seznam (který není navržen pro rychlý přímý přístup k prvkům, ale obsahuje podstatně obecnější skupinu metod).
  - poskytuje optimalizovaný sekvenční přístup
  - nízký stupeň zatížení při operacích vkládání a odstraňování prvků uprostřed seznamu
  - relativně pomalý při přístupu jednotlivým prvkům

Příklad 4 - zásobník, zřetězený seznamu

```
public class StackL {

  private LinkedList stack = new LinkedList();

  public void push(Object o) {
    stack.addFirst(o);
  }

  public Object top() {
    return stack.getFirst();
  }

  public Object pop() {
    return stack.removeFirst();
  }
  .
  .
  .
```

Příklad 4 - zásobník zřetězeného seznamu

```
  .
  .
  .
  public static void main(String[] args) {
    StackL stack = new StackL();

    for(int i = 0; i < 10; i++)
      stack.push("" + i);

    System.out.print("[" + stack.top() + ", ");
    System.out.print(stack.top() + ", ");
    System.out.print(stack.pop() + ", ");
    System.out.print(stack.pop() + ", ");
    System.out.print(stack.pop() + "]");
  }
}
```

Výstup

```
[9, 9, 9, 8, 7]
```

Rozhraní Set

Set (rozhraní)

- má stejné rozhraní jako `Collection`, takže neobsahuje žádné další funkce. Má pouze jiné chování.
- každý prvek vložený do kolekce typu `Set` musí být jedinečný
- všechny objekty vložené do kontejneru musí definovat metodu `equals()` s jejíž pomocí lze určit jedinečnost hodnoty
- rozhraní nezaručuje že bude zachováno nějaké pořadí

HashSet

- rychlé vyhledávání prvků
- objekty musí definovat metodu `hashCode()`

TreeSet

- setříděný kontejner implementovaný pomocí stromové stuktury

Příklad 5 - vytvoření množiny

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

  Set names = new HashSet();
  names.add("Petr");
  names.add("Karel");
  names.add("Hanka");

  names.remove("Karel");
  System.out.println(names.size()); // 2

  names.add("Petr");  //pokus o opětovné přidání Petra
  System.out.println(names.size()); // 2

  System.out.println(names.contains("Hanka")); // true
  System.out.println(names.contains("Karel")); // false
}
```

Rozhraní SortedSet

- jediným dostupným kontejnerem typu `SortedSet` je `TreeSet`
- je zaručeno správné třídění prvků, a proto může být kontejner doplněn o další funkce rozhraní `SortedSet`
  - `Comparator comparator()` - vrátí objekt typu `Comparator`, používaný aktuálním kontejnerem typu `Set`, nebo `null` (přirozené třídění)
  - `Object first()` - vrací nejmenší prvek
  - `Object last()` - vrací největší prvek
  - `SortedSet subSet(fromElement, toElement)` - vrátí podmnožinu určené množiny (včetně)
  - `SortedSet headSet(toElement)` - vrací podmnožinu prvků menších než prvek `toElement`
  - `SortedSet tailSet(fromElement)` - vrací podmnožinu prvků větších nebo rovných prvku `fromElement`

Příklad 6 - třídění

```
public class SortedSetDemo {

  public static void main(String[] args) {
    SortedSet ss = new TreeSet();

    ss.add(new Clovek("Josef", "Vykoukal"));
    ss.add(new Clovek("Dalimil", "Brabec"));
    ss.add(new Clovek("Viktor", "Kotrba"));

    for(Iterator i = ss.iterator(); i.hasNext(); )
      ((Clovek) i.next()).vypisInfo();
  }

}
```

Příklad 6 - třídění

```
class Clovek implements Comparable {
  String jmeno, prijmeni;
  Clovek(String j, String p) {
    jmeno = j;
    prijmeni = p;
  }

  public void vypisInfo() {
    System.out.println(jmeno + " " + prijmeni);
  }

  public int compareTo(Object o) {
    if (o instanceof Clovek) {
      Clovek c = (Clovek) o;
      return prijmeni.compareTo(c.prijmeni);
    }
    else
      throw new IllegalArgumentException("...");
  }
```

Příklad 6 - výstup

```
Dalimil Brabec
Viktor Kotrba
Josef Vykoukal
```

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

Rozhraní Map

Map (rozhraní)

- dvojice klíč-hodnota

HashMap

- implementace založena na hašovací tabulce
- konstantní výkon při vkládání a vyhledávání položek
- v konstruktoru lze nastvit kapacitu a součinitel zatížení hašovací tabulky

TreeMap

- implementace založena na Red-Black Tree
- výsledky zobrazí setříděny (implementace rozhraní `Comparable` nebo `Comparator`)
- jedinou mapou obsahující metodu `subMap()`

Příklad 7 - vytvoření mapy

```
class Counter {
  int i = 1;

  public String toString() {
    return Integer.toString(i);
  }
}
```

Příklad 7 - vytvoření mapy

```
public class Statistics {

  public static void main(String[] args) {
    HashMap hm = new HashMap();
    for(int i = 0; i < 10000; i++) {
      // cisla mezi 0 - 20
      Integer r = new Integer((int) (Math.random() * 20));
      if (hm.containsKey(r))
        ((Counter)hm.get(r)).i++;
      else
        hm.put(r, new Counter());
    }
      System.out.println(hm);
  }
}
```

Příklad 7 - vytvoření mapy

Výstup potom může vypadat např. následovně:

```
{15=522, 4=481, 19=492, 8=485, 11=522, 16=518, 18=484, 3=521,
 7=515, 12=449, 17=500, 2=505, 13=491, 9=515, 6=533, 1=470,
 14=470, 10=517, 5=503, 0=507}
```

Rozhraní SortedMap

- jediným dostupným kontejnerem typu `SortedMap` je `TreeMap`
- je zaručeno správné třídění prvků, a proto může být kontejner doplněn o další funkce rozhraní `SortedMap`
  - `Comparator comparator()` - vytváří objekt typu `Comparator`, používaný aktuálním kontejnerem typu `Map`, nebo `null` (přirozené třídění)
  - `Object first()` - vrací nejmenší klíč
  - `Object last()` - vrací největší klíč
  - `SortedMap subMap(fromKey, toKey)` - vrátí podmnožinu určené mapy (včetně)
  - `SortedMap headMap(toKey)` - vrací podmnožinu klíčů menších než klíč `toKey`
  - `SortedMap tailMap(fromKey)` - vrací podmnožinu klíčů větších nebo rovných klíči `fromKey`

Příklad 8 - uspořádaná mapa

```
public class SortedMapComparatorDemo {

  public static void main(String[]  args) {
    SortedMap sm = new TreeMap(new ClovekComparator());

    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);
    .
    .
    .
```

Příklad 8 - uspořádaná mapa

```
        .
        .
        .
        // 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");
        }
    }
}
```

Příklad 8 - uspořádaná mapa

```
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("...");
  }
}
```

Výstup:

```
Clovek Dalimil Brabec je majitelem uctu se zustatkem 50.0 Kc
Clovek Viktor Kotrba je majitelem uctu se zustatkem 2000.0 Kc
Clovek Josef Vykoukal je majitelem uctu se zustatkem 100.0 Kc
```

Hašování hašový kód

- K vygenerování kontrolního součtu klíče se používá metody `hashCode()` (viz. 15/IV)
  - nezapomenout překrýt i metodu `equals()` (viz. 14/IV)

Příklad s porovnáním zůstatku a majitele účtu.

Faktory ovlivňující výkon struktury HashMap

- Kapacita - počet bloků v tabulce
- Implicitní kapacita - počet bloků po vytvoření tabulky.
  - Kontejnery `HashMap` a `HashSet` obsahují konstruktory, které umožňují implicitní kapacitu určit.
- Velikost - aktuální počet položek v tabulce
- Součinitel zatížení - poměr velikost/kapacita. 0 znamená prázdnou tabulku. 0,5 polovinu tabulky atd.
  - jakmile je této hodnoty dosaženo, zvýší kontejner automaticky svoji kapacitu přibližně o dvojnásobek a přerozdělí existující objekty
  - vyšší hodnota součinitele zatížení snižuje požadavky na paměťový prostor, avšak zvyšuje zatížení systému při vyhledávání záznamů
  - implicitní zatížení pro kontejner `HashMap` je 0,75
  - Kontejnery `HashMap` a `HashSet` mají také konstruktory, které umožňují také určit součinitel zatížení

Starší třídy

- 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()`

Volba implementace List, Set, Map

- Rozdíly mezi jednotlivými kontejnery závisí na způsobu implementace, jsou to datové struktury, které fyzicky implementují požadované rozhraní.
- Nejpřesvědčivějším způsobem, jímž se můžeme přesvědčit o rozdílech mezi jednotlivými implementacemi, je sledování výkonu.

Volba druhu seznamu

Přiložený kód vytvoří vnitřní bázovou třídu, kterou budeme používat coby testovací aplikační rámec. Potom vytvoří po jednom poli anonymních vnitřních tříd pro každý test. Každá z těchto vnitřních tříd bude volat metodu `test()`.

Typ	Získat	Procházet	Vložit	Vyjmout
Pole	141	468	Nelze	Nelze
ArrayList	250	1265	328	28861
LinkedList	7844	1000	109	16
Vector	281	1484	312	30501

Volba mezi kontejnery typu Set

U množin můžete volit mezi třídami `TreeSet` a `HashSet`, a to podle velikosti objektu `Set` (potřebujete-li setříděný výstup, použijte třídu `TreeSet`).

Typ	Velikost testu	Přidat	Obsahuje	Procházet
TreeSet	10	20,30	7,80	35,90
	100	28,13	22,65	42,96
	1000	23,82	26,70	8,78
HashSet	10	9,40	4,70	32,8
	100	9,53	14,53	42,19
	1000	7,55	16,73	8,14
LinkedHashSet	10	12,50	3,20	32,80
	100	11,25	15,62	36,56
	1000	7,64	16,59	5,27

Volba mezi typy Map

Výkon mapy je velmi významně ovlivněn také její velikostí.

Typ	Velikost testu	Přidat	Obsahuje	Procházet
TreeMap	10	23,40	7,80	35,90
	100	29,85	21,41	42,50
	1000	24,08	25,02	8,19
HashMap	10	11,00	3,10	31,3
	100	10,15	14,22	41,10
	1000	7,46	15,95	8,02
LinkedHashMap	10	12,50	4,70	31,20
	100	13,13	14,37	25,32
	1000	8,36	16,42	4,91
HashTable	10	10,90	4,70	53,10
	100	9,69	14,84	45,01
	1000	8,05	18,05	9,42

Nástroje

- Řazení a prohledávání seznamů
  - Nástroje pro řazení a prohledávání seznamů mají stejné názvy a signatury jako nástroje pro řazení pole, jsou to však statické metody třídy `Collections`.
- Třída `Collections` obsahuje celou řadu nástrojů
  - třídění, transformace, ...
  - vytváření neměnných kolekcí a map
  - synchronizaci kolekcí a map

Collections - třídění, transformace, ...

- `binarySearch(List, Object)` - binární vyhledávánní (nutné předtím zavolat `sort`)
- `copy(List, List)`
- `enumeration(Collection)` - vrací starý objekt `Enumeration`
- `fill(List, Object)` - nahrazuje všechny prvky seznamu daným objektem
- `max(Collection)`, `max(Collection)` - největší/nejmenší prvek (přirozené třídění)
- `nCopies(int, Object)` - vrací seznam délky n, jehož odkazy ukazují na daný objekt
- `reverse()` - obrací pořadí prvků v seznamu

Collections - neměnné kolekce a mapy

- vytvoří kolekci nebo mapu s přístupem pouze pro čtení
  - `Collection Collection.unmodifiableCollection(Collection)`
  - `List Collection.unmodifiableList(List)`
  - `Set Collection.unmodifiableSet(Set)`
  - `Map Collection.unmodifiableMap(Map)`
- při pokusu o změnu obsahu vyhodí vyjímku `UnsupportedOperationException`

Příklad 9 - vytvoření neměnné kolekce

```
public class UnmodifiableDemo {

  public static void main(String[] args) {
    Collection c = new ArrayList();
    c.add("Objekt");
    //je treba naplnit pred zmenou na nemennou kolekci
    c = Collections.unmodifiableCollection(c);
    System.out.println(c); //cteni ok
    c.add("Dalsi objekt"); //UnsupportedOperationException
  }

}
```

Collections - synchronizace kolekcí a map

- součást **multithreadingu**
- nový kontejner ihned po vytvoření "protáhnout" příslušnou metodou
  - `Collection Collection.synchronizedCollection(Collection)`
  - `List Collection.synchronizedList(List)`
  - `Set Collection.synchronizedSet(Set)`
  - `Map Collection.synchronizedMap(Map)`

Například

```
Collection c = Collections.synchronizedCollection(
  new ArrayList());
```

Souběžný přístup

- Zamezení souběžného vykonávání více než jednoho procesu
  - vkládání, mazání, modifikace jiným procesem
  - změna velikosti kontejneru až po zavolání `size()`
  - ...
- Vyhazuje vyjímku `ConcurrentModificationException`
- Někdy nazýváno jako mechanizmus **rychlého selhání**

Příklad 10 - souběžný přístup

```
public static void main(String[] args) {
  Collection c = new ArrayList();
  Iterator it = c.iterator();
  c.add("Objekt");
  String s = (String) it.next(); //vyvolá vyjímku
}
```

Proč?

- K vyjímce dojde proto, že je něco vloženo až **po** získání iterátoru
- Při testování přístupu pomocí pomocí metody `get()` mechnizmus selhává