Kontejnery obecně, rozhraní `Collection`

Table of Contents

Kontejnery (dynamické datové struktury)

Co jsou kontejnery, kolekce (`Collection`)?

- *Dynamické datové struktury* vhodné k ukládání proměnného počtu objektů (přesněji odkazů na objekty).
- Primitivní hodnoty se do základních dynamických struktur neukládají přímo, ale prostřednictvím objektových obálek (wrappers).
- Jsou automaticky ukládané v operační paměti (ne na disku).

Proč je používat?

- pro uchování **proměnného počtu** objektů (počet prvků se může měnit — zvyšovat, snižovat)
- oproti polím nabízejí efektivnější algoritmy přístupu k prvkům
- přístupem rozumíme *vložení*, *smazání*, či *nalezení* prvku

Kvalitní seznámení s kontejnery najdete na stránkach Oracle

Připomenutí polí v Pythonu

- Pole je v Pythonu použitelné pro kompaktní ukládání primitivních hodnot (čísla, znaky…), ale ne objektů

```
array('l')
array('u', 'hello \u2641')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.14])
```

Připomenutí struktur v Pythonu

Seznamy

`my_list = ["a", "b", "mpilgrim", "z", "example"]` *modifikovatelné*

N-tice

`my_tuple = ("a", "b", "mpilgrim", "z", "example")` *nemodifikovatelné*

Množiny

`my_set = {'a', False, 'b', True, 'mpilgrim', 42}` *modifikovatelné*

Slovníky

asociativní pole, `my_dict = {'server': 'db.diveintopython3.org', 'database': 'mysql'}` *modifikovatelné*

Typ prvků nás nezajímá, mohou být namíchané.

Připomenutí seznamů v Pythonu

- Datové typy v Pythonu

```
# create a 5-item list
a_list = ['a', 'b', 'mpilgrim', 'z', 'example']
# take 3rd item
m = a_list[2] # 'mpilgrim'
```

```
# take last item (at length-1)
m = a_list[-1] # 'example'
# make a slice (výřez)
s = a_list[:2] # ['a', 'b', 'mpilgrim']
```

Jak to bude v Javě?

- Principiálně podobně, v detailu odlišně
- Seznam vytvoříme jako jiné objekty pomocí `new`
- Kapacita vytvořeného seznamu není omezená
- Pak teprve lze přidávat prvky
- Neměnné seznamy lze i `List.of("Jedna", "Dva")`
- Přístup k prvku metodou: `myList.get(2)` vrátí 3. prvek
- Nelze použít závorky pro index: `myList[2]`
- Obdobně u množin a asociativních polí, tedy slovníků, map

Základní kategorie

Základní kategorie jsou dány tím, které **rozhraní** daný kontejner implementuje:

Seznam (`List<E>`)
: lineární struktura, každý prvek má svůj číselný index (pozici)

Množina (`Set<E>`)
: struktura bez duplicitních hodnot a (obecně) bez uspořádání

Mapa, slovník, asociativní pole (`Map<K,V>`)
: struktura uchovávající dvojice (klíč→hodnota), rychlý přístup přes klíč

Ke všem těmti rozhraním Java nabízí hotové implementace (i několik), případně si implementujeme vlastní.

Typové parametry

- V Javě byly dříve kontejnery koncipovány jako *beztypové*,
- do jednoho bylo možné ukládat prvky různých typů - lidi, řetězce, cokoli.
- Nyní mají *typové parametry* ve špičatých závorkách (např. `Set<Person>`), které
- určují, jaký typ položek se do kontejneru smí dostat.

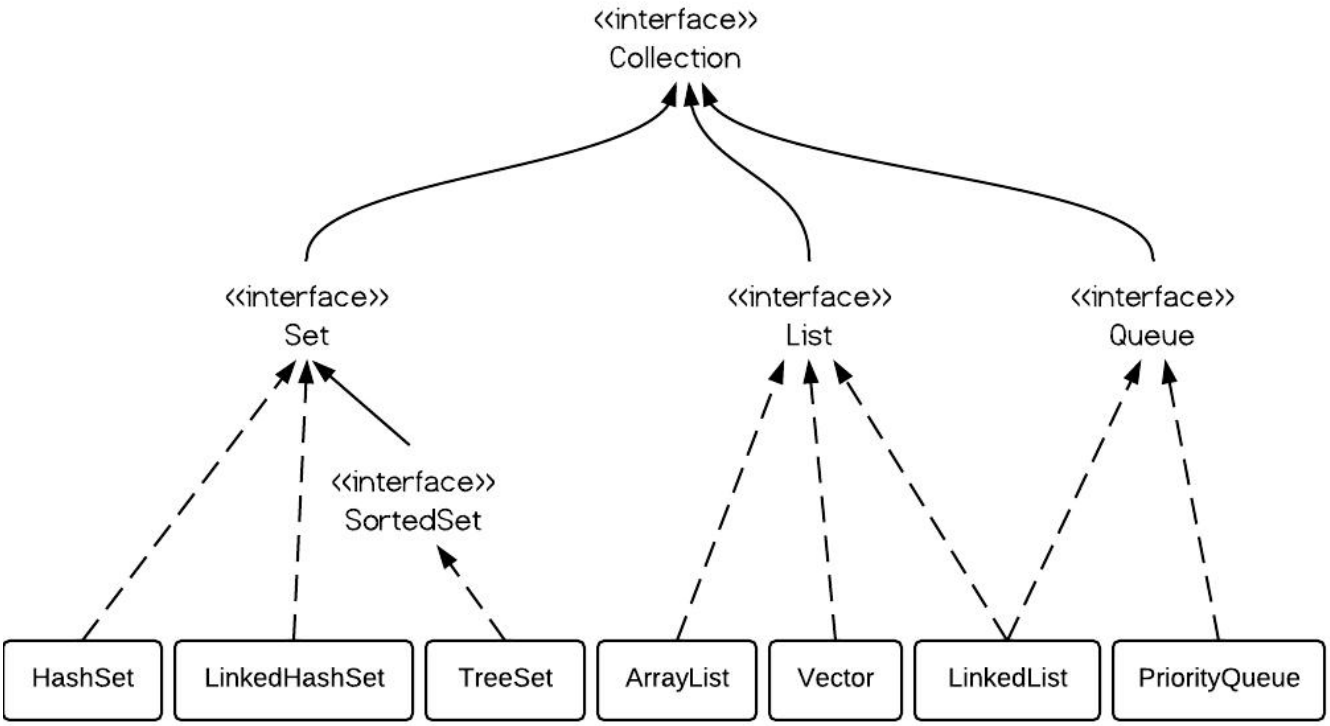
```
List objects; // without type, old, BAD
```

```
List<String> strings; // with type String - OK
```

Kontejnery bez typů nepoužívat! (Vždy používejte špičaté závorky.)

Typová hierarchie kolekcí

- Třídy jsou součástí *Java Core API* (balík `java.util`).



Rozhraní `Collection`, `List`, `Set`

`Collection`
: je nejobecnější rozhraní, zahrnuje cokoli, co shromažďuje jednotlivé prvky; dokumentace API: `Collection`

`List`, `Set`
: specifická rozšíření rozhraní `Collection`

ne však `Map` - asociativní pole není `Collection`!

Třídy implementující tato rozhraní

- `ArrayList`, `LinkedList`, `HashSet`, ...

Příklady

```
List<String> listOfStrings = new ArrayList<>();
// new empty set of strings
Collection<String> collection = new HashSet<>();
// new unmodifiable list of 2 strings
List<String> listWithValues = List.of("so", "cool");
```

Metody přidání a zjištění obsahu

Kolekce prvků typu E (tj. `Collection<E>`) má následující metody:

`boolean add(E e)`

přidá prvek `e` do kolekce, `true` jestli se skutečne přidal (využití u množin)

`int size()`

vrátí počet prvků v kolekci

`boolean isEmpty()`

`true` je-li kolekce prázdná (velikost je 0)

Příklady

```
Collection<String> set = new HashSet<>();
set.add("Pacman"); // true
set.add("Pacman"); // false
set.toString(); //  ["Pacman"]

set.size(); // 1
set.isEmpty(); // false
```

Metody odebrání a dotazu na přítomnost

`void clear()`

odstraní všechny prvky z kolekce

`boolean contains(Object o)`

`true`, právě když se `o` v kolekci nachází; na test rovnosti se použije `equals`

`boolean remove(Object o)`

odstraní prvek z kolekce; vrátí `true`, byl-li prvek odstraněn

Příklady

```
List<String> list = List.of("Hello", "world");
list.toString(); //  ["Hello", "world"]

list.contains("Hello"); // true
list.remove("Hello");   // true

list.toString(); //  ["world"]
list.contains("Hello"); // false

list.clear();
list.toString(); // [] empty list
```

Iterátor z kolekce

`Iterator<E> iterator()`

metoda každé kolekce; vrací něco, přes co se dá kolekce iterovat (procházet for-each cyklem)

- Jedná se o použití návrhového vzoru Iterator:
- Kolekce nenabízí přímo metody pro procházení. Místo toho vrací objekt (iterátor), který umožňuje danou kolekci procházet jednotným způsobem bez ohledu na to, jak je kolekce uvnitř implementovaná.
- Je možná i varianta, kdy procházení je umožněno jak specifickými metodami, tak iterátorem. Příkladem je `List`, který nabízí metodu `get(int i)` pro přímé procházení i obecný iterátor z důvodu kompatibility s ostatními kolekcemi.

Převod kolekce na pole

`T[] toArray(T[] a)`

vrátí z kolekce pole typu `T`, tj. je to konverze kolekce na pole

```
Collection<String> c = List.of("a", "b", "c");
String[] stringArray = c.toArray(new String[0]);
// stringArray contains "a", "b", "c" elements
```

Metody pro hromadné manipulace

`boolean containsAll(Collection<?> c)`

`true` právě když kolekce obsahuje všechny prvky z `c`

Metody vracející `true`, když byla kolekce změněna:

`boolean addAll(Collection<E> c)`

přidá do kolekce všechny prvky z `c`

`boolean removeAll(Collection<?> c)`

udělá rozdíl kolekcí (`this - c`)

`boolean retainAll(Collection<?> c)`

udělá průnik kolekcí

Výraz `Collection<?>` reprezentuje kolekci objektů jakéhokoliv typu.

Příklad metod `Collection` V

```
Collection<String> c1 = List.of("A", "A", "B");
Collection<String> c2 = Set.of("A", "B", "C");
c1.containsAll(c2); // false
c2.containsAll(c1); // true

c1.retainAll(c2); // true
// c1 is ["A", "B"]

c1.removeAll(c2); // true
// c1 is empty []

c1.addAll(c2); // true
// c1 contains elements ["A", "B", "C"]
```

Repl.it demo ke kontejnerům

- https://replit.com/@tpitner/PB162-Java-Lecture-07-collections