Kontejnery obecně, rozhraní `Collection`

```
<link rel="stylesheet"  href="http://cdnjs.cloudflare.com/ajax/libs/font-awesome/3.1.0/css/font-awesome.min.css">
```

Kontejnery (dynamické datové struktury)

Co jsou kontejnery, kolekce (collections)?

- *dynamické datové struktury* vhodné k ukládání proměnného počtu objektů (přesněji odkazů na objekty)
- jsou automaticky ukládané v operační paměti (ne na disk)

Proč je používat?

- pro uchování **proměnného počtu** objektů (počet prvků se může měnit — zvyšovat, snižovat)
- oproti polím nabízejí efektivnější algoritmy přístupu k prvkům

Kvalitní seznámení s kontejnery najdete na stránkach Oracle

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

Vlastní implementace se dá vytvořit implementováním zmíněných rozhraní.

Typové parametry

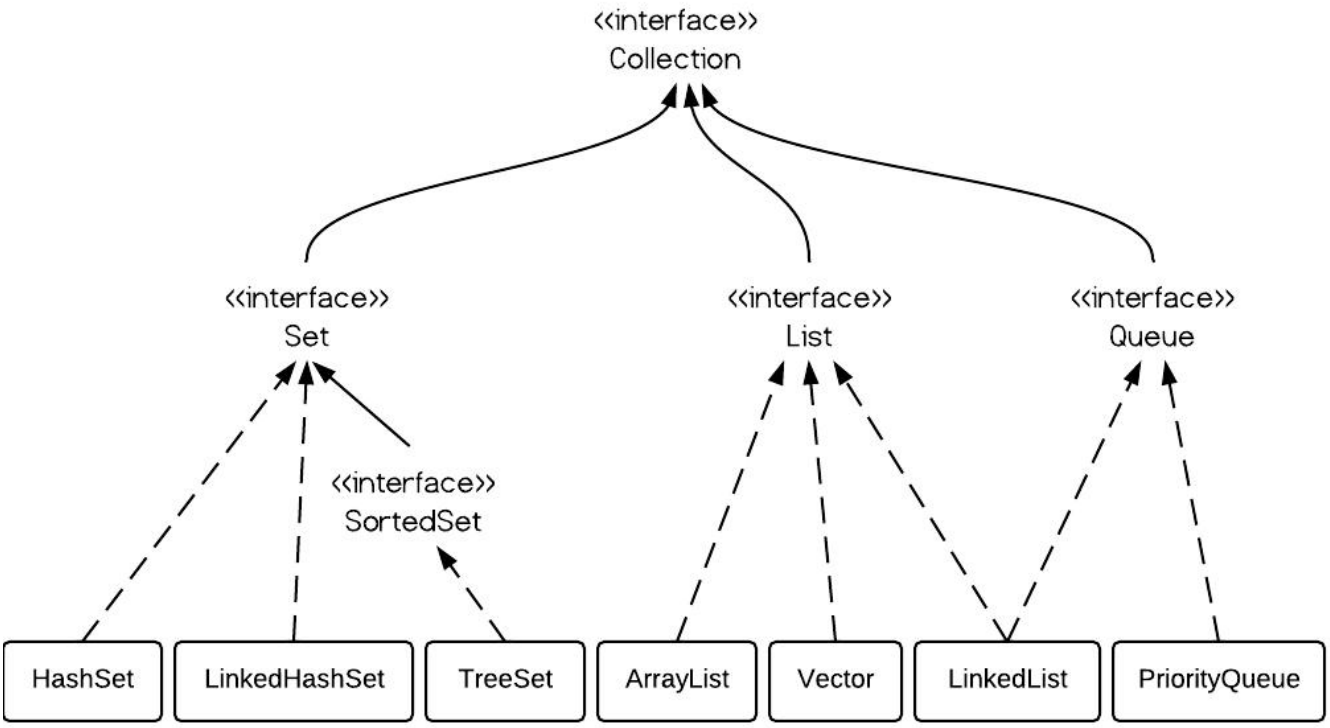
- v Javě byly dříve kontejnery koncipovány jako *beztypové*
- teď mají *typové parametry* ve špičatých závorkách (např. `Set<Person>`)
- určují, jaký typ položek se do kontejneru smí dostat

```
List objects; // without type, old!
List<String> strings; // with type String
```

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

Hierarchie kolekcí

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



`Collection`

- Mezi třídy typu rozhraní `Collection` patří:
  - rozhraní `List, Set,`... (ne však `Map`!)
  - konkrétní implementace `ArrayList, LinkedList, HashSet,`...
- Vytvoření **prázdné** kolekce:

```
List<String> listOfStrings = new ArrayList<>();
Collection<String> collection = new HashSet<>();
List<String> listWithValues = List.of("so", "cool");
```

Metody `Collection` I

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` jestli je kolekce prázdná (velikost je 0)

Příklad metod `Collection` I

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

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

Metody `Collection` II

- `void clear()`
  - odstraní všechny prvky z kolekce
- `boolean contains(Object o)`
  - `true` jestli se `o` nachází v kolekci
  - na test rovnosti se použije `equals`
- `boolean remove(Object o)`
  - odstraní prvek z kolekce
  - `true` jestli byl prvek odstraněn

Příklad metod `Collection` II

```
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(); //  []
```

Metody `Collection` III

- `Iterator<E> iterator()`
  - vrací něco, co se dá 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.

Metody `Collection` IV

- `T[] toArray(T[] a)`
  - vrátí pole typu `T`
  - konverze kolekce na pole
  - použití:

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

Metody `Collection` V

- `boolean containsAll(Collection<?> c)`
  - `true` jestli 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 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 ["A", "B"]

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

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

Repl.it demo ke kontejnerům

- https://repl.it/@tpitner/PB162-Java-Lecture-07-collections