Práce s kolekcemi – algoritmy a implementace ve zvoleném programovacím jazyce.

„Kolekce jsou objekty, které seskupují skupinu elementů do jednoho celku (=kontejnery)“

Další zdroje vhodné pro prostudování:
Kontejnery I. - Úvod do problematiky, Základy práce s kolekcemi v jazyce Java
Kontejnery II. - Kategorie kontejnerových rozhraní, Implementace, Wrapppery
Kontejnery III. – Algoritmy

Kontejnery (v Javě obvykle nazývané "kolekce") jsou datové objekty, které nám umožňují snadno a efektivně spravovat variabilní hromadná data (i další objekty). Podle způsobů uložení dat a práce s nimi rozlišujeme různé druhy kontejnerů a při jejich použití si vždy vybereme ten, který se pro daný případ nejlépe hodí v konkrétním jazyce. Základními operacemi dle typu kontejneru je iterace, porovnávání, přidávání a mazání.

Zvolený jazyk: Java

- **Pole** – nejzákladnější struktura vhodná zejména pro primitivní hodnoty, které se nemusí nahrazovat objektem (známe předem počet objektů ve skupině). Nevýhodou je jeho dynamické plnění (*nutné definovat jeho rozsah – časová náročnost*). Pole mohou být vícerozměrné.
- **Kolekce** – struktura vhodná zejména pro práci s objekty, kdy se „nemusíme“ starat o velikost kolekce. Nové objekty lze vkládat bez nutnosti realokace. Vhodné při vytváření předem neznámého počtu objektů.

The Collection Framework (CF) – systém rozhraní a tříd, které bezprostředně souvisí s kolekcemi. Naprostou většinu z nich najdeme v balíku java.util (v Javě 5.0 - tj. od JDK 1.5.0 došlo v CF k významným změnám).

Související pojmy:

- **Iterátor** – prostředek zajišťující sekvenční přístup k datům, pracující krokově, v každém dalším kroku poskytne přístup k dalšímu prvku. Rychlé právě v případě sekvenčního procházení. *Zajišťuje možnost procházení prvků bez znalosti jejich implementace*.
    - nezáleží na pořadí, nebo je pořadí dané vlastnostmi kolekce
    - metoda *iterator()* daného kontejneru
    - metoda *next()* - vrátí další prvek v kontejneru
    - metoda *hasNext()* - je-li v kontejneru další prvek
    - metoda *remove()* - vyjmutí posledního objektu vráceného iterátorem
- **Porovnatelnost** – důležitá vlastnost, kterou potřebujeme pro uložení do některých kontejnerů. Datové objekty mohou mít porovnatelnost implementovánu prakticky libovolným způsobem, třída implementuje rozhraní Comparable (obsahuje jedinou metodu `compareTo()`).
    - **Vector** – nejzákladnější implementace = „pole“ s proměnnou velikostí indexace od 0, v případě uložení primitivního datového typu nutné zapouzdření. Třída ještě před vznikem CF. Chování třídy `Vector` je prakticky shodné s třídou `ArrayList` s drobnými rozdíly. `Vector` má některé metody navíc (např. hledání od určitého indexu), lze mu stanovit přírůstek kapacity a je synchronizovaný (viz dále). V běžných případech dnes nemáme důvod používat `Vector` namísto třídy `ArrayList`.
    - **HashTable** – implementuje hašovací tabulku mapující klíče na hodnoty (obojí obecně typu Object) -> polymorfní, každý objekt - klíč musí implementovat metody hashCode a equals

Kolekce – základní „operace“

add(object) – boolean *addAll(Collection)* *clear()* *contains(Object*) – boolean *containsAll(Collection)* – boolean *isEmpty()* - boolean *iterator()* - vrací iterátor *remove(Object)* *removeAll(Collection)* – boolean *retainAll(Collection)* – ponechá ty, které se schodují s argumentem *size()* *toArray()* - vrátí pole	addAll(kolekce,pozice) get(pozice) indexOf(objekt) lastIndexOf(objekt) remove(pozice) listIterator(), listIterator(pozice) set(pozice, objekt) subList(od,do)

Máme dvě základní množiny kontejnerů, každá z nich vychází z jednoho rozhraní:

- **Kolekce** – „normální kolekce“ - skupina samostatných prvků (rozhraní *Collection*)
  Vždy zahrnují:
  - Rozhraní: umožňují pracovat s kolekcí bez ohledu na detaily její implementace
  - Implementaci: konkrétní znovupoužitelné implementace rozhraní.
  - Algoritmy: metody, které provádějí užitečné výpočty typu vyhledávání, třídění s objekty kolekcí

  - **Množiny** (rozhraní `Set`) - každý prvek může být v kontejneru **pouze jednou** (jinak kontejner zabrání vložení). Není zde obdoba kontejneru *multiset* (možnost vícenásobné přítomnosti téhož prvku). Prvek obecně nemá určenu žádnou polohu v množině, na základě které by k němu bylo možné přistupovat. Zvláštním případem je podrozhraní `SortedSet`, což je seřazená množina.

    - objekty, vložené do kontejneru, musí mít definovat metodu *equals()* - umožní určit jedinečnost

    Implementace

    - *HashSet*
      - množina s hešovací tabulkou -V situaci, kdy je třeba uložit obrovské pole nějakých řetězců a jednotlivé řetězce velmi rychle vyhledávat. Sekvenční prohledávání díky své velké časové náročnosti není vhodné. Jednotlivé řetězce se rozdělilí do skupin, které si jsou podobné. K tomu jsou hešovací funkce, která dokážou jednotlivé řetězce rozptýlit do oněch políček.
      - Objekty musí definovat metodu *hashCode()* - vrací celé číslo (int) „co nejlépe“ charakterizující obsah objektu
    - *TreeSet*
      - setříděný kontejner typu množina
      - rozhraní *SortedSet*

        - metody *first()* - vrátí nejmenší prvek
        - *last()* - největší prvek
        - *subSet(fromElement, toElement)* – podmnožina od určeného do druhého určeného prvku
        - *headSet(toElement)* – podmnožina menších prvků, než *toElement*
        - *tailSet(fromElement)* – podmnožina větších prvků, než *fromElement*

  - **Seznamy** (rozhraní `List`) - prvek může být v kontejneru vícekrát a má určenu jednoznačnou polohu (index). Indexován od 0.
    - seřazení seznamu – *sort(list)*

- promíchání seznamu – *shuffle(list)*
- obrácení pořadí – *reverse(list)*
- hledání binárním dělením, kopírování seznamu, konverze kolekcí na pole atd.

Implementace

- *ArrayList*

```
ArrayList<Person> persons = new ArrayList();
```

  - seznam implementovaný pomocí pole
  - rychlý přímý přístup k prvkům
  - vkládání a odebírání je pomalé!
- *LinkedList*
  - vkládání a odstraňování prvků je optimalizované, stejně tak sekvenční procházení
  - přímý přístup k prvkům je relativně pomalý
  - navíc obsahuje metody jako *addFirst(), addLast(), getFirst(), getLast(), removeFirst(), removeLast()*
  - nejsou definovány v rozhraní
    - umožňují zřetězený seznam použít jako zásobník, frontu či oboustrannou frontu

- **Mapa – „**asociativní kolekce (pole)“ - (mapované = obsahující dvojice klíč-hodnota), rozhraní *Map*. Lze rovněž rozšiřovat do více rozměrů => vytvoří se mapa, jejímiž hodnotami jsou také mapy. Objekty tedy nevyhledáváme podle čísla (indexu), ale pomocí jiného objektu - klíče.
Každý člen této dvojice může být libovolného referenčního typu (rozhraní, objektové třídy a pole, přistupujeme v zásadě pouze pomocí referencí). Neobsahuje duplicitní prvky.

  - metoda *put(klíč, hodnota)* vloží nejen hodnotu, ale zároveň její klíč
  - metoda *get(klíč)* – vrátí objekt odpovídající danému klíč

  Implementace

  - *HashMap*
    - založeno na hešovací tabulce
    - rychlé vkládání a vyhledávání položek
  - *TreeMap*
    - oproti *HashMap* přináší seřazení klíčů
    - výsledky se zobrazují setříděné

Nevýhody kontejnerů:

- Neznámý typ
  - kontejner obsahuje odkazy na objekty obecného typu *Object* (kořenová třída všech objektů)
  - neexistuje žádné omezení pro to, co se dá do kontejneru uložit (ani v případě, kdyby si to člověk výslovně přál.
    Od verze Javy 5.0 (JDK 1.5.x): `Vector<String> vector = new Vector<String>();`
  - je-li třeba objekt uložený v kontejneru použít, je nutné přetypovat na správný typ. Primitivní prvky je třeba vkládat jako zapouzdřující referenční typ.