Колекције

Све колекцијске класе дефинисане су у пакету **java.util** и представљају генеричке типове
(постоји параметар или параметри који одређују ког типа су елементи колекције).
Последица тога је да елементи колекције могу бити произвољног класног или интерфејсног типа, а у случају да желимо да у колекцији чувамо вредности примитивног типа, за тип елемената колекције треба навести омотач-класу тог примитивног типа.

Колекције се користе у ситуацијама када не знамо унапред колико ћемо елемената имати или нам је потребно више флексибилности у начину на који приступамо елементу колекције него што имамо индексирањем низа.

Колекције генерално имају могућност да се шире како би се прилагодиле потребном броју елемената.

На вежбама радимо само делић колекција расположивих у Јавиној библиотеци:

- **Iterator<T>** интерфејсни тип, декларише методе за итерирање кроз елементе колекције, један по један
  ListIterator<T>
- **ArrayList<T>** тип, низолика структура за смештање произвољног типа објеката. Број објеката које можемо
  (Vector <T>) сместити аутоматски се повећава по потреби. (проширив низ)
- **LinkedList <T>** тип, подржава смештање произвољног типа објеката у двоструко повезану листу
- **HashMap <K,V>** тип, подржава смештање објеката типа **V** у хеш табелу коју називамо каталогом/мапом.
  Објекат се смешта коришћењем придруженог кључа, који је објекат типа **K**.
  Да бисмо приступили објекту, морамо проследити његов кључ.

Колекција је објекат који представља скуп објеката датог типа, груписаних заједно и организованих на одређени начин у меморији. **Када причамо о колекцијама објеката, заправо мислимо на колекције референци на објекте**: колекције садрже само референце на објекте, док су сами објекти смештени негде другде у меморији (екстерни су за колекцију).

Постоје три основна типа колекција за организовање елемената на разне начине:
скупови (sets), низови (sequences) и мапе/каталози (maps)

Скупови нису уређени и у њима нема понављања елемената - сваки објекат у скупу мора бити јединствен.
Постоје и уређене варијанте скупова.
Основне операције над скупом су:

- додавање елемента
- итерирање кроз све елементе
- провера да ли је дати објекат елемент скупа
- уклањање елемента на који имамо референцу.

У **низове** се објекти смештају линеарно, а могу садржати и више копија истог
објекта. Основне операције над низом су:

- додавање елемента на почетак, на крај или након датог објекта у низу,
- приступ елементу - првом, последњем или на задатој позицији (индексирање низа)
- тражење објекта једнаког датом (провером свих елемената низа било унапред или уназад)
- уклањање елемента - првог, последњег, са дате позиције или једнаког датом објекту.

Сваки објекат смештен у **мапу** има придружен објекат кључ и објекат и његов кључ се смештају заједно, у пару.
Кључ одређује где ће пар бити смештен у мапи, а када желимо да приступимо објекту, морамо проследити одговарајући кључ. Сви кључеви у мапи морају бити различити јер кључ мора јединствено да идентификује придружени објекат.

Итератори

Итератор је објекат који можемо да користимо једанпут за приступ свим објектима колекције, једном по једном. Коришћење итератора је стандардни механизам за приступ свим елементима колекције.
За сваку колекцију која представља скуп или низ (од оних које радимо на вежбама: **ArrayList<>** , **LinkedList<>**) могуће је позивом метода **iterator()** добити референцу на одговарајући објекат итератор за текући садржај колекције, док колекције које представљају мапе немају методе за креирање итератора.

Интерфејс **Iterator<>** (из пакета **java.util**) декларише следећа 3 метода (која се могу позивати за објекат итератор):

- **T next()** враћа објекат типа ***T***, почев од првог и поставља објекат ***Iterator<T>*** тако да врати следећи објекат приликом следећег позива овог метода. Ако нема објекта који треба вратити, метод избацује изузетак типа ***NoSuchElementException***
- **boolean hasNext()** враћа ***true*** ако постоји објекат коме ће се приступити методом ***next()***, ***false*** иначе
- **void remove()** уклања последњи објекат враћен методом ***next()*** из колекције. Ако ***next()*** није позван или позовемо ***remove()*** 2 пута након позива ***next()***, избацује се изузетак типа ***IllegalStateException*** Не подржавају сви итератори овај метод и уколико га позовемо у том случају, биће избачен изузетак типа ***UnsupportedOperationException***

Пролазак кроз колекцију помоћу итератора, типично се врши у петљи следећег облика:

```
MojaKlasa element;
while( iter.hasNext() ) {
        element = iter.next();
        // radimo nesto sa tekucim elementom kolekcije: “element” …
}
```

Овде се претпоставља да је **iter** типа **Iterator<MojaKlasa>** и да је добијен позивом метода **iterator()** за одговарајућу колекцију објеката типа **MojaKlasa**, кроз коју се пролази у горњој петљи.

Сваки пут када изнова треба проћи кроз објекте колекције, потребан је нови итератор, пошто је итератор објекат за „једнократну употребу“. Итератор је „једносмерна улица“ - можемо проћи кроз објекте колекције, 1 по 1, једном и то је то.

Ако то није довољно, постоји флексибилнија врста итератора - ***list iterator***.

Лист итератори

Интерфејс ***ListIterator<>***, дефинисан у пакету **java.util**, декларише методе за кретање кроз колекцију унапред и уназад, па се једном објекту може приступити и више од једанпут.
Интерфејс ***ListIterator<>*** наслеђује интерфејс ***Iterator<>***, те се могу примењивати и методи суперинтерфејса које смо управо видели.

Методи дефинисани у интерфејсу ***ListIterator<>*** за кретање кроз листу објеката су:

- **next()** — као код ***Iterator<>***
- **boolean hasNext()** — као код ***Iterator<>***
- **int nextIndex()** — враћа индекс објекта који ће бити враћен следећим позивом ***next()*** као тип ***int***, или враћа број елемената у листи ако је објекат ***ListIterator<>*** на крају листе
- **T previous()** — враћа претходни објекат, користи се за кретање уназад кроз листу
- **boolean hasPrevious()** — враћа ***true*** ако ће следећи позив ***previous()*** вратити објекат
- **int previousIndex()** — враћа индекс објекта који ће бити враћен следећим позивом ***previous()*** или **-1** ако је објекат ***ListIterator<>*** на почетку листе

Позиви метода **next()** и **previous()** могу се преплитати. Позив **previous()** непосредно након позива **next()**, и обрнуто, вратиће исти елемент.

ListIterator<> методи за додавање, уклањање и замену објекта колекције су:

- **void remove()** — уклања последњи објекат коме је приступљено помоћу ***next()*** или ***previous()*** (***UnsupportedOperationException***, ***IllegalStateException***)
- **void add(T obj)** — додаје аргумент непосредно испред објекта који би био враћен наредним позивом ***next()*** или иза објекта који би био враћен наредним позивом ***previous()***. Позив ***next()*** након операције ***add()*** вратиће додати објекат. Наредни позив ***previous()*** није измењен овом операцијом (***UnsupportedOperationException***, ***ClassCastException***, ***IllegalOperationException***)
- **void set(T obj)** — мења последњи објекат коме је приступљено позивом **next()** или **previous()** (***IllegalStateException***, ***UnsupportedOperationException***, ***ClassCastException***, ***IllegalArgumentException***)

За колекције чије класе имплементирају интерфејс **List<>** (од оних које радимо на вежбама: **ArrayList<>**, **LinkedList<>**) референцу на објекат који представља лист итератор могуће је добити позивом метода **listiterator()**:

- **ListIterator<T> listIterator()** — враћа лист итератор за текући садржај листе
- **ListIterator<T> listIterator(int index)** — враћа лист итератор за текући садржај листе почев од задате позиције. Задати индекс одређује први елемент који би био враћен иницијалним позивом **next()**. Иницијални позив **previous()** вратио би елемент чији је индекс за један мањи од задатог. (***IndexOutOfBoundsException***)