JAVA programavimo kalba

Vidinės klasės (nested classes) ir iteratorius (Iterator)

Vidinė klasė (inner class)

Java kalboje galima aprašyti klasę kitos klasės viduje

```
class A {

    class B {

    }

}
```

Vidinė klasė (inner class)

Tokios vidinės klasės gali būti dviejų tipų - statinės (pažymėtos static) ir ne.

```
class A {
    static class B {
    }
}
```

Vidinė klasė – statinė (static inner class)

Iš statinės vidinės klasės negalima prieiti prie paprastų išorinės klasės metodų, tik prie statinių. Tokia vidinė statinė klasė iš išorės prieinama per išorinę klasę:

- *Išorinė-klasė.vidinė-klasė*
- ClassA.InnerClassB a = new ClassA.InnerClassB()

Vidinė klasė – paprasta (inner class)

Paprastos (ne statinės) vidinės klasės turi priėjimą prie išorinės klasės laukų ir metodų net jei jie yra privatūs, t.y. vidinės klasės metodai turi tokias pat teises kaip ir išorinės klasės metodai. Jei iš vidinės klasės norime prieiti prie išorinės klasės laukų ar metodų, tai reikia naudoti konstrukcją:

- *Išorinė-klasė***.this.***išorinės-klasės-laukas-ar-metodas*
- ClassA**.this.**methodA()
- ClassA**.this.**fieldB = 3;

Lokali klasė (local class)

Klasė gali būti aprašyta ir bet kokio bloko viduje. Blokas - tai java sakiniai apgaubti riestiniais skliaustais: { … }

```
if (a > 100)  {
    class LocalA {
    }

    LocalA b = new LocalA(…);
}
```

Anoniminė klasė (anonymous class)

Anoniminė klasė tai klasė, kuri aprašoma išraiškoje, kur reikalingas objektas pagal interfeisą arba abstrakčią klasę:

```
interface A {
        long pow2();
}

A a = new A()  {
        public long pow2() { … }
};
```

Iteratorius (Iterator)

Kolekcijos (ir bet kokios klasės), kurių elementus galima ištraukti vieną po kito tam tikra tvarka, turi realizuoti Iterable<Type> sąsają. Šis interfeisas reikalauja realizuoti tik vieną metodą iterator(), kuris turi grąžinti iteratoriaus objektą, t.y. objektą, kuris realizuoja sąsają Iterator<Type>

- Iterator interfeisas reikalauja realizuoti šiuos metodus:

```
boolean hasNext();
Type next();
void remove();
```

Pastaba: metodą remove() neprivalu realizuoti, jei kolekcija nepalaiko bet kokio elemento naikinimo.

Iteratorius (iterator)

- Standartinis ciklas naudojant iteratorių:

```
Iterator<Type> iterator = collection.iterator();
while (iterator.hasNext()) {
        Type s = iterator.next();
}
```

```
for (Type s : collection) {

};
```

Iteratorius

Ciklo metu negalima trinti ar pridėti elementų į iteruojamą kolekciją:

```
for (Type s : collection) {
        s.add(a);
        s.remove(b);
};
```

Iteratorius

Jei ciklo metu norime trinti elementus tai reikia tiesiogiai naudoti iteratorių:

```
Iterator<Type> iterator = collection.iterator();
while (iterator.hasNext()) {
    Type s = iterator.next();

    iterator.remove();
};
```

Sinchronizuotos kolekcijos (synchronized collections)

Visos iki šios aptartos kolekcijos (List, Set, SortedSet, Map, SortedMap) yra nepritaikytos dirbti daugiagijame (multithread) režime. Egzistuoja kolekcijų klasės apvalkalai (wrapper) skirtos valdyti kolekcijas daugiagijame režime. Tokios kolekcijos sukuriamos per taip vadinamus metodus-fabrikus (t.y. tokie metodai, kurie sukuria ir grąžina naują objektą):

- List<Type> list = Collections.synchronizedList(new ArrayList<>());
- Set<Type> set = Collections.synchronizedSet(new HashSet<>());
- SortedSet<Type> set = Collections.synchronizedSortedSet(new TreeSet<>());
- Map<Key, Value> map = Collections.synchronizedMap(new HashMap<>());
- SortedMap<Key, Value> map = Collections.synchronizedSortedMap(new TreeMap<>());
- Collection<Type> collection = Collections.synchronizedCollection(new ArrayList<>());

Kolekcijų funkcijos (collections functions)

Kolekcijų karkasas (framework) susideda ne tik iš interfeisų ir klasių, bet ir iš funkcijų - java.util.Collections klasės statinių metodų.

- Labai daug visokių metodų darbui su kolekcijomis:

lygiavimas, paieška, max ar min elementų paieška ir t.t.

Lyginimo interfeisas (Comparable ir Comparator)

- Comparator<Type> interface turi metoda compare(Type a, Type b) ir nusako pagal ką bus lyginami objektai.
  - class SortByNameSurname implements Comparator<Student>

- Taip pat galime implementuoti Comparable pačioje klasėje :
  - class Student implements Comparable<Student>

Uždaviniai

1. Tarkime turime klasę Employee ir jos vidinę klasę Address. Tegul klasėje Employee turime du laukus - String name ir Address address, o Address klasė turi laukus - String city, String address. Sukurkite keletą darbuotojų su vardu ir adresu ir pabandykit nustatyti iš kiek skirtingų miestų yra darbuotojai. Pvz:
    1. Aleksas – Vilnius,
    2. Martinas – Siauliai,
    3. Egle – Druskininkai.
    4. Atsakymas bus: 3 skirtingi miestai, [Vilnius, Siauliai, Druskininkai]
2. Sukurkite automobilių klasę su laukais: numeris, marke, savininkas (vardas, pavarde). Išveskite automobilius sulygiuotus pagal savininko pavardę, vardą ir automobilio numerį. Lygiuokite naudodami java.util.Collections.sort();