05 - Iterator

Credits: Askar Gafurov

Uzitocne materialy:

- Slajdy z prednasky
- Poznamky z minulych rokov
- Head First Design Patterns, chapter 9

Iterator

Pouzivame, ked chceme prejsť (preiterovat) cez vsetky prvky nejakeho zlozeneho objektu (napriklad Composite) bez toho, aby sme odhalovali vnutornu strukturu objektu.

Pouzitie:

Standardne rozhranie `Iterator` Javy umoznuje iterovat cez standardny for-loop:

```
// a simplified version of the standard `Iterator` interface
// type `T` shows that the iterator returns objects of type `T`

public interface Iterator<T> {
    boolean hasNext(); // whether it has something to iterate
    T next() throws NoSuchElementException; // get next object
    // if no object is left, then throw NoSuchElementException
}
```

Standardne rozhranie pre iterovatelne objekty je rozhranie `Iterable`:

```
public interface Iterable<T> {
    Iterator<T> iterator(); // return a new iterator of type `T`
}
```

Takto vyzera iterovanie cez objekt s rozhranim `Iterable` pomocou for-loop:

```
ArrayList<Integer> list = new ArrayList<>(Arrays.asList(1,2,3,4,5));
// `ArrayList<T>` implements `Iterable<T>` interface

for (Integer element: list) {
    foo(element);
}
```

Ako to funguje pod kapotou:

```
static <T> void forEach(Iterable<T> iterable) {
    Iterator<T> iterator = iterable.iterator();

    while (iterator.hasNext()) {
        final T element = iterator.next();
        foo(element);
    }
}
…
forEach(list);
```

Zakladna struktura:

Kedze standardne rozhrania su uz v Jave implementovane, treba len doplnit potrebne metody pre splnenie rozhrania:

```
public class MyArrayList implements Iterable<Integer> {
    private Integer[] elements;
    public MyArrayList(Integer ... elements) {
        this.elements = elements.clone();
    }

    @Override
    public Iterator<Integer> iterator() {
        return new MyArrayListIterator(this);
    }

    // sic! Inner class
    private class MyArrayListIterator implements Iterator<Integer> {
        private final MyArrayList list;
        private int nextIndex;

        private MyArrayListIterator(MyArrayList list) {
            this.list = list;
            this.nextIndex = 0;
        }

        @Override
        public boolean hasNext() {
            return nextIndex < list.elements.length;
        }

        @Override
        public Integer next() {
            if (hasNext()) {
                final Integer value = list.elements[nextIndex];
                nextIndex++;
                return value;
            }
```

```
            else {
                throw new NoSuchElementException();
            }
        }
    }
}
```

Ulohy:

U: Stromovu strukturu obohatte o rozhranie `Iterable<TreeNode>` (vid Kostra implementacie stromu) pre iterovanie cez vsetky vrcholy v pre-order poradi (t.j. najprv vypisat koren, potom lavy podstom rekurzivne a nasledne pravy podstrom rekurzivne). Vyskusajte vypisat identifikatory vrcholov pomocou for-cyklu aj while-cyklu.

U*: Pridajte metodu `Iterator<TreeNode> orderedIterator()`, ktory vrati iterator pre vrcholy tak, aby ich identifikatory mali vzostupne poradie. Vyskusajte ho pomocou while-cyklu.

U**: Implementujte metodu `orderedIterator()` z predoslej ulohy ako `default` metodu rozhrania `TreeNode` (skuste to spravit na 4 riadky alebo menej).

U: Implementujte iterovatelnu triedu `Range` s konstruktorom `Range(int upperBound)`, ktorej iterator bude vracat postupne cele cisla v intervale <0, upperBound) (prava hranica otvorena).
Priklad pouzitia:

```
for(Integer i: new Range(10)) {
     System.out.println(i);
}
// will print numbers from 0 to 9 (inclusive), one number per line
```

U: Pridajte do triedy `Range` konstruktor `Range(int lowerBound, int upperBound)`, po pouziti ktoreho iterator bude vracat cisla v intervale <lowerBound, upperBound).

U: Pridajte do triedy `Range` konstruktor `Range(int lowerBound, int upperBound, int step)`, po pouziti ktoreho iterator bude vracat cisla postupne lowerBound, lowerBound+step, lowerBound+2*step, … kym su mensie nez upperBound.

U: Implementujte iterovatelnu triedu s konstruktorom `InfiniteRange(int start)`, ktorej iterator bude postupne vracat cele cisla, pocnuc cislom start.

U: Pridajte do triedy `InfiniteRange` konstruktor `InfiniteRange(int start, int step)` s analogickou funkcionalitou.

U*: Vytvorte dekorator `SkipN` rozhrania Iterable<T>, ktory donuti iterator dekorovaneho objektu vracat kazdy N-ty prvok, pricom parameter N dostane dekorator ako argument

konstruktora. Zaistite, aby dekorator pouzival konstantne mnozstvo pamate. Vyskusajte ho na objekte triedy `ArrayList<String>`.

U*: Vytvorte dekorator `Slice` rozhrania Iterable, ktory bude fungovat analogicky s triedou `Range`, t.j. dostane na vstupe Iterable `it` a premenne lowerBound, upperBound, step a pri iterovani vrati postupne prvky `it[lowerBound]`, `it[lowerBound+step]`, `it[lowerBound+2*step]`, etc.

U**: Vytvorte dekorator `LowPassFilter` rozhrania Iterable, ktore donuti iterator dekorovaneho objektu vracat iba prvky nizsie nez prahova hodnota `threshold`, pricom hodnotu `threshold` dostane dekorator ako argument konstruktora. Zaistite, aby dekorator pouzial konstantne mnozstvo pamate. Hint: pouzite rozhranie `Comparable<T>`.

U**: Vytvorte triedu `Permutations`, implementujucu rozhranie `Iterable<List<Integer>>`, ktorej iterator bude iterovat cez vsetky permutacie cisel 1 az N, kde N dostane ako parameter konstruktora.

U**: Vytvorte triedu `Permute`, implementujucu rozhranie `Iterable<List<T>>`, ktorej iterator bude iterovat cez vsetky permutacie zoznamu, ktory dostane ako parameter konstruktora.

Pomocky:

Kostra implementacie stromu:

```
public interface TreeNode {
    int getId();
}

public class LeafNode implements TreeNode {
    private final int id;

    public LeafNode(int id) {
        this.id = id;
    }
    @Override
    public int getId() {
        return id;
    }
}

public class InternalNode implements TreeNode {
    private final TreeNode left, right;
    private final int id;
```

```
    public InternalNode(int id, TreeNode left, TreeNode right) {
        this.id = id;
        this.left = left;
        this.right = right;
    }

    @Override
    public int getId() {
        return id;
    }
}
```