Лабораторная работа №4
«Реализация итераторов в языке Java»

Скоробогатов С.Ю.

7 апреля 2016 г.

1 Цель работы

Изучение обобщённых итераторов и экземплярных вложенных классов языка Java.

2 Исходные данные

2.1 Интерфейсы Iterator и Iterable

Обобщённый интерфейс java.util.Iterator является контрактом, которому должны удовлетворять классы, объекты которых предназначены для перебора элементов некоторого множества значений:

```
public interface Iterator<E> {
        boolean hasNext();
        E next();
}
```

Объекты классов, реализующих этот интерфейс, называются *итераторами*. Тип перебираемых значений задаётся типовым параметром E, метод hasNext итератора возвращает true, если ещё остались нерассмотренные значения, а метод next возвращает следующее значение.

Использование итератора можно проиллюстрировать следующим примером. Пусть в переменной it находится ссылка на итератор, перебирающий объекты некоторого класса SomeType. Тогда перебор всех объектов можно организовать в цикле такого вида:

```
while (it.hasNext()) {
        SomeType x = it.next();
        // Сделать что-то с объектом x
}
```

Контейнерные классы, как правило, реализуют обобщённый интерфейс Iterable, в котором объявлен метод iterator. Этот метод предназначен для создания нового итератора для перебора объектов, содержащихся в контейнере (т.е. в объекте контейнерного класса):

```
public interface Iterable<T> {
        Iterator<T> iterator();
}
```

Разрешено создавать сразу несколько итераторов для одного контейнера. Эти итераторы работают совершенно независимо, что позволяет, например, реализовать на двух итераторах перебор всех пар объектов, содержащихся в контейнере:

```
Iterator<SomeType> i = container.iterator();
while (i.hasNext()) {
        SomeType a = i.next();
        Iterator<SomeType> j = container.iterator();
        while (j.hasNext()) {
                SomeType b = j.next();
                // Сделать что-то с парой (a, b)
        }
}
```

2.2 Специальная форма оператора for

Контейнер, класс которого реализует интерфейс Iterable, можно использовать в специальной форме оператора for:

```
for (тип_элемента переменная : контейнер) ...
```

Эта форма оператора for является сокращённой записью следующего фрагмента кода:

```
Iterator<тип_элемента> it = контейнер.iterator();
while (it.hasNext()) {
        тип_элемента переменная = it.next();
        ...
}
```

Тем самым, перебор всех пар объектов контейнера можно переписать как

```
for (SomeType a : container) {
        for (SomeType b : container) {
                // Сделать что-то с парой (a, b)
        }
}
```

Для общности специальную форму оператора for также разрешено использовать для перебора элементов массива.

2.3 Реализация итераторов через вложенные классы

Как правило, итератору необходим доступ к внутреннему состоянию контейнера. Чтобы не нарушать инкапсуляцию, удобно реализовать итератор в виде экземплярного вложенного класса внутри контейнерного класса.

В качестве примера рассмотрим класс SuffixList, представляющий список суффиксов изменяемой строки. Изменяемые строки в Java представляются классом StringBuilder. Тем самым, объект класса SuffixList будет контейнером для единственного объекта класса StringBuilder и будет предоставлять итератор по суффиксам строки, хранящейся в этом объекте.

Внутри класса SuffixList мы объявим вложенный экземплярный класс SuffixIterator, в поле pos которого будет храниться индекс первого символа следующего суффикса. При создании итератора в поле pos будет записываться 0. Значение поля pos будет увеличиваться на единицу при каждом вызове метода next.

```
import java.util.Iterator;

public class SuffixList implements Iterable<String> {
        private StringBuilder s;

        public SuffixList(StringBuilder s) { this.s = s; }

        public Iterator<String> iterator() { return new SuffixIterator(); }

        private class SuffixIterator implements Iterator<String> {
                private int pos;

                public SuffixIterator() { pos = 0; }

                public boolean hasNext() { return pos < s.length(); }

                public String next() {
                        return s.substring(pos++, s.length());
                }
        }
}
```

Для демонстрации работоспособности класса SuffixList создадим класс Test:

```
public class Test {
        public static void main(String[] args) {
                StringBuilder b = new StringBuilder("qwerty");
                SuffixList suff = new SuffixList(b);
                for (String s : suff) System.out.println(s);

                b.insert(1, 'x');
                for (String s : suff) System.out.println(s);
        }
}
```

3 Задание

Во время выполнения лабораторной работы требуется разработать на языке Java один из классов, перечисленных в таблицах 1 и 2. Класс должен реализовывать интерфейс Iterable<T>.

Объект разрабатывемого класса должен быть изменяемым, то есть в нём надо так или иначе предусмотреть возможность изменения внутреннего состояния.

В методе main вспомогательного класса Test нужно продемонстрировать работоспособность разработанного класса.

Таблица 1: Варианты классов

1	Последовательность целых чисел с итератором по цифрам десятичного представления чисел.
2	Множество целых чисел с итератором по размещениям с повторениями по m элементов (m задаётся в конструкторе).
3	Изменяемая строка с итератором по всем непустым подстрокам.
4	Последовательность строк с итератором по количествам различных общих букв в двух соседних строках.
5	Последовтельность точек в трёхмерном пространстве с итератором по длинам отрезков, соединяющих соседние точки.
6	Полином с итератором по его производным.
7	Последовательность целых чисел с итератором по наибольшим общим делителям соседних чисел последовательности.
8	Множество целых чисел с итератором по размещениям без повторений по m элементов (m задаётся в конструкторе).
9	Строка с итератором по словам (слова в строке разделены произвольным количеством пробелов).
10	Последовательность строк с итератором по строкам, являющимся подстрокой следующей строки в последовательности.
11	Ломаная линия на плоскости с итератором по векторам нормалей к составляющим её отрезкам.
12	Последовательность нормализованных дробей с итератором по суммам соседних дробей.
13	Последовательность целых чисел с итератором по подпоследовательностям, сумма элементов которых не превышает 21 (подпоследовательности имеют максимально возможную длину и не пересекаются).
14	Множество целых чисел с итератором по сочетаниям по m элементов (m задаётся в конструкторе).
15	Изменяемая строка с итератором по содержащимся в ней латинским гласным буквам.
16	Последовательность строк, состоящих из разделённых пробелами слов, с итератором по подпоследовательностям, слова которых могут поместиться в строку из 80 символов (подпоследовательности имеют максимально возможную длину и не пересекаются).
17	Последовательность n-мерных векторов с итератором по скалярным произведениям соседних векторов.
18	Окружность с итератором по k точкам, равномерно распределённым по её длине.
19	Последовательность целых чисел с итератором по ненулевым суммам трёх соседних элементов.
20	Множество целых чисел с итератором по наименьшим общим кратным всех непустых подмножеств.
21	Изменяемая строка с итератором по индексам первых букв вхождений заданной подстроки w (w задаётся в конструкторе).
22	Последовательность строк с итератором по максимальным суффиксам, совпадающим с префиксом следующей строки последовательности.
23	Последовательность трёхмерных векторов с итератором по векторным произведениям соседних векторов.
24	Параметризованная типовым параметром T последовательность контейнеров, классы которых реализуют интерфейс Iterable<T> (итератор – «конкатенация» итераторов контейнеров).

Таблица 2: Варианты классов

25	Изменяемая строка с итератором по префиксам, имеющим грань длины k (k задаётся в конструкторе).
26	Последовательность целых чисел с итератором по степеням двойки, присутствующим в последовательности.
27	Простой неориентированный граф, представленный матрицей смежности, с итератором по вершинам в порядке обхода в глубину.
28	Множество точек на плоскости с итератором, перебирающим точки в порядке возрастания длины их радиус-вектора (при реализации итератора использовать очередь с приоритетом).
29	Матрица размера $m \times n$ с итератором по всем возможным подматрицам (подматрица получается из матрицы удалением произвольных строк и/или столбцов).
30	Простой неориентированный граф, представленный матрицей смежности, с итератором по вершинам в порядке обхода в ширину.
31	Обобщённое бинарное несбалансированное дерево поиска (ключи реализуют интерфейс Comparable, значения – любые) с итератором по словарным парам в порядке возрастания ключей.
32	Обобщённый кольцевой буфер с итератором по элементам представляемой им очереди (элементы должны перебираться в том порядке, в каком они добавлялись в очередь).
33	Обобщённый однонаправленный связанный список с итератором по значениям, хранящимся в его элементах.
34	Простой неориентированный граф, представленный списками инцидентности, с итератором по вершинам в порядке обхода в глубину.
35	Целочисленная матрица размера $m \times n$ с итератором по суммам элементов строк.
36	Множество отрезков на плоскости с итератором по всем точкам пересечения этих отрезков.
37	Последовательность целых чисел с итератором по всем соседним парам чисел.
38.	Бинарное отношение на множестве целых чисел от 0 до n с итератором по всем парам чисел, принадлежащим отношению (отношение должно быть представлено булевской матрицей).
39.	Последовательность булевских значений размера n с итератором по элементам последовательности (последовательность должна быть представлена массивом байтов, по восемь булевских значений на байт).
40	Простой неориентированный граф, представленный списками инцидентности, с итератором по вершинам в порядке обхода в ширину.
41.	Предложение, состоящее из разделённых пробелами слов, с итератором по словам, являющимся изображениями целых чисел в десятичной системе счесления.
42.	Булевская матрица размером $m \times n$, где $1 \leq m, n \leq 8$, с итератором по суммам элементов строк по модулю 2 (т.е., исключающее ИЛИ). Элементы матрицы должны быть закодированы битами в числе типа **long**.