Лабораторная работа № 6

Тема: Использование класса коллекций

Источник:

Брюс Эккель. Философия Java.

McGraw, Kathy Sierra, Bert Bates. SCJP Sun certified programmer for Java 6 study guide

Библиотека утилит Java (java.util.*) содержит большой набор классов коллекций/контейнеров. Контейнеры обладают разнообразными возможностями для работы с объектами и их и хранения, с помощью их удаётся решить огромное количество задач.

Основные концепции

В Java для хранения объектов предоставляется два интерфейса:

- Коллекция. Класс List (список) хранит элементы в порядке вставки. Класс Set (множество) хранит не повторяющиеся элементы. Класс Queue (очередь) выдаёт элементы в порядке, определяемом спецификой очереди (обычно это порядок вставки элементов в очередь).
- Карта. Класс Map (карта/словарь/ассоциативный массив) хранит наборы пар объектов «ключ=значение», с возможностью выбора значения по ключу.

Коллекции в Java различаются тем, сколько в одной ячейке коллекции «помещается» элементов. Коллекции (Collection) содержат только один элемент в каждой ячейке. В коллекции Map (карта) хранятся два объекта: ключ и связанное с ним значение.

Коллекции ArrayList и LinkedList принадлежат к семейству List. Элементы в этих коллекциях хранятся в порядке добавления. Однако они различаются не только скоростью выполнения тех или иных операций, но и количеством операций (LinkedList содержит больше операций, чем ArrayList).

В множествах Set (HashSet, TreeSet и LinkedHashSet) каждый элемент хранится только в одном экземпляре, а разные реализации Set используют разный порядок хранения элементов. В HashSet порядок элементов определяется по значению функции хеширования хранимого в контейнере класса, хотя на первый взгляд порядок следования элементов будет выглядит хаотично. В TreeSet объекты хранятся отсортированными в зависимости от условия 2сравнения. В LinkedHashSet обеспечивается хранение элементов в порядке добавления.

Коллекция Map позволяет искать объекты по ключу, как несложная база данных. Объект, ассоциированный с ключом, называется значением.

В идеале весь программный код должен писаться в расчёте на взаимодействие с этими интерфейсами, а точный тип указываться только в точке создания. Объект List может быть создан как:

List<Apple> apples = new ArrayList<Apple>();

Если позднее возникнет необходимость изменить реализацию, достаточно сделать это в точке создания:

List<Apple> apples = new LinkedList<Apple>();

Такой подход работает не всегда, потому что некоторые классы обладают дополнительной функциональностью. Например, LinkedList содержит дополнительные методы, не входящие в интерфейс List, а TreeMap — методы, не входящие в Map. Если такие методы используются в программе, восходящее преобразование к обобщённому интерфейсу невозможно.

Добавление групп элементов в коллекцию

Для включения групп элементов в коллекцию, классы Arrays и Collections содержат вспомогательные методы. Метод Arrays.asList() получает либо массив, либо список элементов, разделённых запятыми, и преобразует его в объект List. Метод Collections.addAll получает объект Collection и либо массив, либо список, разделённый запятыми, и добавляет элементы в Collection.

```
import java.util.*;
import java.io.*;
class AddingGroups {
    public static PrintStream out = System.out;
    public static void main(String...args) {
        List<Integer> list_0 = Arrays.asList(1, 2, 3, 4, 5);
        List<Integer> list_1 = new ArrayList<Integer>(Arrays.asList(6, 7, 8, 9, 10));
        Integer[] array_0 = new Integer[]{11, 12, 13, 14, 15};

        Collection<Integer> collection = new ArrayList<Integer>();
        out.println(collection);

        collection = new ArrayList<Integer>(list_0);
        out.println(collection);

        collection.addAll(list_1);
        out.println(collection);

        Collections.addAll(collection, array_0);
        out.println(collection);

        Collections.addAll(collection, 16, 17, 18, 19, 20);
        out.println(collection);
```

```
    }
}
```

Вывод содержимого контейнеров

Для получения печатного представления массива необходимо использовать метод Arrays.toString, но контейнеры отлично выводятся и без посторонней помощи. Следующий пример демонстрирует использование основных типов контейнеров:

```
import java.util.*;
import java.io.*;
class PrintingContainers {
    public static PrintStream out = System.out;
    public static Collection fill(Collection<String> collection) {
        collection.add("cat");
        collection.add("dog");
        collection.add("rat");
        collection.add("rat");
        return collection;
    }
    public static Map fill(Map<String, String> map) {
        map.put("cat", "Fuzzy");
        map.put("dog", "Rags");
        map.put("rat", "Bosco");
        map.put("rat", "Spot");
        return map;
    }
    public static void main(String...args) {
        out.println(fill(new ArrayList<String>()));
        out.println(fill(new LinkedList<String>()));
        out.println(fill(new HashSet<String>()));
        out.println(fill(new TreeSet<String>()));
        out.println(fill(new LinkedHashSet<String>()));
        out.println(fill(new HashMap<String, String>()));
        out.println(fill(new TreeMap<String, String>()));
        out.println(fill(new LinkedHashMap<String, String>()));
    }
}
```

Список

Существует две основные разновидности коллекции List: ArrayList и LinkedList. ArrayList оптимизирована для произвольного доступа к элементам, однако операции вставки/удаления элементов в середину списка работают относительно медленно. LinkedList, оптимизирована для последовательного доступа к элементам, с быстрыми операциями вставки (удаления) в середину списка. Произвольный доступ к элементам Linked List выполняется относительно медленно, но по широте возможностей он превосходит ArrayList.

LinkedList

LinkedList реализует базовый интерфейс List. Класс LinkedList также содержит методы, позволяющие использовать его в качестве стека, очереди (Queue) или двухсторонней очереди (дека).

PМетоды getFirst() и element() идентичны — они возвращают начало (первый элемент) списка без его удаления и выдают исключение NoSuchElementException для пустого списка. Метод peek() возвращает первый элемент списка или null для пустого списка. Метод addFirst() вставляет элемент в начало списка. Метод offer() делает то же, что add() и addLast() — он добавляет элемент в конец списка. Метод removeLast() удаляет и возвращает последний элемент списка.

Следующий пример демонстрирует схожие и различающиеся аспекты этих методов:

```
import java.util.*;
import java.io.*;
class LinkedListFeatures {
	public static PrintStream out = System.out;
	public static void main(String...args) {
		LinkedList<String> words = new LinkedList<String>();
		Collections.addAll(words, "one", "two", "three", "four", "five", "six", "seven", "eight",
"nine", "ten");
		out.println(words);

		out.println(words.getFirst() + " " + words.element() + " " + words.peek());
		out.println(words.remove() + " " + words.removeFirst() + " " + words.poll());
		out.println(words.removeLast());

		words.addFirst("0");
		words.add("10");
		words.addLast("11");
		words.offer("12");
```

```
            out.println(words);
        }
    }
```

Итераторы

Итератор — это объект, обеспечивающий перемещение по последовательности объектов с выбором каждого объекта этой последовательности, без углубления в её структуру. Вдобавок, итератор является «легковесным» (lightweight) объектом: его создание обходиться без заметных затрат ресурсов. Из-за этого итераторы часто имеют ограничения; например, Iterator в Java поддерживает перемещение только в одном направлении, он предоставляет возможность:

- запросить у контейнера итератор вызовом метода iterator(). Полученный итератор готов вернуть начальный элемент последовательности при первом вызове своего метода next().
- получить следующий элемент последовательности вызовом метода next().
- проверить, остались ли еще объекты в последовательности (метод hasNext()).
- удалить из последовательности последний элемент, возвращённый итератором, методом remove().

```
import java.util.*;
import java.io.*;

class SimpleIteration {
    public static PrintStream out = System.out;
    public static void main(String...args) {
        List<String> words = Arrays.asList("The weather is wonderfull today !".split(" "));
        Iterator<String> iterator = words.iterator();

        while(iterator.hasNext()) {
            out.print("\"" + iterator.next() + "\"" + " " );
        }
        out.println();

        for (iterator = words.iterator(); iterator.hasNext(); iterator.next()) {
            out.print(iterator.next() + " - ");
        }
        out.println();

        for (String word: words) {
            out.print(word + "#");
```

```
        }
        out.println();
    }
}
```

При использовании Iterator можно не беспокоиться о количестве элементов в последовательности. Проверка осуществляется методами hasNext() и next(). При переборе элементов списка в одном направлении, без его модификации, можно использовать более компактную запись - «синтаксис foreach».

ListIterator

ListIterator — более мощная разновидность Iterator, поддерживаемая только классами List. Если Iterator поддерживает перемещение только вперёд, List-Iterator является двусторонним. Кроме того, он может выдавать индексы следующего и предыдущего элементов по отношению к текущей позиции итератора в списке и заменять последний посещённый элемент методом set(). Вызов listIterator() возвращает ListIterator, указывающий в начало List, а для создания итератора ListIterator, изначально установленного на элемент с индексом n, используется вызов listiterator(n). Все перечисленные возможности продемонстрированы в следующем примере:

```
import java.util.*;
import java.io.*;
class ListIteration {
    public static PrintStream out = System.out;
    public static void main(String...args) {
        List<Integer> numbers = Arrays.asList(0, 1, 2, 3, 4, 5, 6, 7);
        out.println(numbers);

        ListIterator<Integer> it = numbers.listIterator();
        while(it.hasNext()) {
            int value = it.next();
            out.println(it.previousIndex() + " " + value + " " + it.nextIndex());
        }

        while(it.hasPrevious()) {
            out.print(it.previous() + " ");
        }
        out.println();
```

```
            it = numbers.listIterator(5);
            while(it.hasNext()) {
                it.next();
                it.set(0);
            }
            out.println(numbers);

        }
}
```

Множество

В множествах (Set) каждое значение может храниться только в одном экземпляре. Попытки добавить новый экземпляр эквивалентного объекта блокируются. Множества часто используются для проверки принадлежности объекта заданному множеству. Следовательно, важнейшей операцией Set является операция поиска, поэтому на практике обычно выбирается реализация HashSet, оптимизированная для быстрого поиска.

Set имеет такой же интерфейс, что и Collection. В сущности, Set и является Collection, но обладает несколько иным поведением (кстати, идеальный пример использования наследования и полиморфизма: выражение разных концепций поведения). В следующем примере в множество HashSet включаются пятьдесят случайных чисел от 0 до 25; однако в результате каждое число присутствует в коллекции только в одном экземпляре.

```
import java.util.*;
import java.io.*;

class SetOfInteger {
    public static PrintStream out = System.out;
    public static void main(String...args) {
        Random rand = new Random(System.nanoTime());
        Set<Integer> set = new HashSet<Integer>();
        for (int i = 0; i < 50; i++) {
            set.add(rand.nextInt(25));
        }
        out.print(set);
    }
}
```

Непредсказуемый порядок следования чисел в выводе объясняется тем, что HashSet использует хеширование для ускорения выборки. Порядок, поддерживаемый HashSet, отличается от порядка TreeSet или LinkedHashSet, поскольку каждая реализация упорядочивает элементы по своему. Если требуется чтобы результат был отсортирован, следует воспользоваться TreeSet вместо HashSet. Одной из наиболее распространённых операций со множествами является проверка принадлежности методом contains(), но существуют и другие операции:

```
import java.util.*;
import java.io.*;
class SetOperations {
	public static PrintStream out = System.out;
	public static void main(String...args) {
		Set<String> setA = new HashSet<String>();
		Collections.addAll(setA, "A B C D E F G H I J K L".split(" "));
		// setA.add("M");
		out.println("Does set A contains \'H\'? - " + setA.contains("H"));
		out.println("Does set A contains \'N\'? - " + setA.contains("N"));

		Set<String> setB = new HashSet<String>();
		Collections.addAll(setB, "H I J K L".split(" "));
		out.println(setA.containsAll(setB));
		setA.removeAll(setB);
		out.println(setA);
	}
}
```

Карта

Возможность отображения одних объектов на другие (ассоциация) чрезвычайно полезна при решении широкого класса задач программирования. В качестве примера рассмотрим программу, анализирующую качество распределения класса Java Random. В идеале класс Random должен выдавать абсолютно равномерное распределение чисел, но чтобы убедиться в этом, необходимо сгенерировать большое количество случайных чисел и подсчитать их количество в разных интервалах. Множества упрощают эту задачу: ключом в данном случае является число, сгенерированное при помощи Random, а значением - количество его вхождений:

```
import java.util.*;
import java.io.*;
class MapExample {
```

```
    public static PrintStream out = System.out;
    public static void main(String...args) {
        Random rand = new Random(System.nanoTime());
        Map<Integer, Integer> m = new HashMap<Integer, Integer>();
        for (int i = 0; i < 10000; i++) {
            int r = rand.nextInt(25);
            Integer freq = m.get(r);
            if (freq == null) {
                m.put(r, 1);
            } else {
                m.put(r, freq + 1);
            }
        }
        out.println("Keys are: " + m.keySet());
        out.println("Values are: " + m.values());
        out.println("Map is:");
        for (Integer key: m.keySet()) {
            out.print(key + " " + m.get(i) + ", ");
        }
        out.println();
    }
}
```

В main() механизм автоматической упаковки преобразует случайно сгенерированное целое число в ссылку на Integer, которая может использоваться с HashMap (контейнеры не могут использоваться для хранения примитивов). Метод get() возвращает null, если элемент отсутствует в контейнере (то есть если число было сгенерировано впервые. В противном случае метод get() возвращает значение Integer, связанное с ключом, и последнее увеличивается на 1 (автоматическая упаковка снова упрощает вычисления, но в действительности при этом выполняются преобразования к Integer и обратно).

Map может вернуть Set своих ключей, Collection значений или множество Set всех пар «ключ=значение». Метод keySet() создает множество всех ключей, которое затем используется в синтаксисе foreach для перебора Map.

Стек

Стек часто называют контейнером, работающим по принципу «первым вошел, последним вышел» (LIFO). То есть элемент, последним занесённый в стек, будет первым, полученным при

извлечении из стека. В классе LinkedList имеются методы, напрямую реализующие функциональность стека.

Очередь

Очередь обычно представляет собой контейнер, работающий по принципу «первым вошел, первым вышел» (FIFO). Иначе говоря, элементы заносятся в очередь с одного «конца» и извлекаются с другого в порядке их поступления. Очереди часто применяются для реализации надежной передачи объектов между разными областями программы.

Класс LinkedList содержит методы, поддерживающие поведение очереди, и реализует интерфейс Queue, поэтому LinkedList может использоваться в качестве реализации Queue.

PriorityQueue

Принцип FIFO описывает наиболее типичную организацию очереди. Именно организация очереди определяет, какой элемент будет следующим для заданного состояния очереди. Правило FIFO означает, что следующим элементом будет тот, который дольше всего находится в очереди.

В приоритетной очереди следующим элементом считается элемент, обладающий наивысшим приоритетом. Например, в аэропорту пассажира, самолет которого скоро улетит, могут пропустить без очереди. В системах обработки сообщений некоторые сообщения могут быть важнее других и должны обрабатываться как можно скорее, независимо от момента их поступления. Параметризованный класс PriorityQueue был добавлен в Java SE5 как механизм автоматической реализации этого поведения.

При помещении объекта в PriorityQueue вызовом offer() объект сортируется в очереди. По умолчанию используется естественный порядок помещения объектов в очередь, однако вы можете изменить его, предоставив собственную реализацию Comparator. PriorityQueue гарантирует, что при вызове peek(), poll() или removeQ вы получите элемент с наивысшим приоритетом.

Создание приоритетной очереди для встроенных типов — Integer, String, Character и т. д. — является делом тривиальным. В следующем примере используются те же значения, что и в предыдущем, но PriorityQueue выдаёт их в другом порядке:

```
import java.util.*;
import java.io.*;

class PriorityQueueDemo {
        public static PrintStream out = System.out;
        public static void print(Queue queue) {
                while(queue.peek() != null) {
```

```
                out.print(queue.remove() + " ");
            }
            out.println();
        }
        public static void main(String...args) {
            Random r = new Random(System.nanoTime());
            PriorityQueue<Integer> priorityQueue = new PriorityQueue<Integer>();
            for (int i = 0; i < 10; i++) {
                priorityQueue.offer(r.nextInt(i + 10));
            }
            out.println(priorityQueue);
            print(priorityQueue);

            List<Integer> integers = Arrays.asList(25, 22, 20, 18, 14, 9, 3, 1, 1, 2, 3, 9, 14, 18, 21,
23, 25);
            priorityQueue = new PriorityQueue<Integer>(integers);
            print(priorityQueue);

            priorityQueue = new PriorityQueue<Integer>(integers.size(), Collections.reverse
Order());
            priorityQueue.addAll(integers);
            print(priorityQueue);

            String string = "Education should eschew obfuscation!";
            List<String> chars = Arrays.asList(string.split(""));
            PriorityQueue<String> pqos = new PriorityQueue<String>(chars);
            print(pqos);

            pqos = new PriorityQueue<String>(chars.size(), Collections.reverseOrder());
            pqos.addAll(chars);
            print(pqos);

            Set<Character> charSet = new HashSet<Character>();
            for (char c: string.toCharArray()) {
                charSet.add(c);
            }
            PriorityQueue<Character> pqoc = new PriorityQueue<Character>(charSet);
            print(pqoc);
        }
    }
```

Дубликаты разрешены, а меньшие значения обладают более высокими приоритетами. Чтобы показать, как изменить порядок элементов посредством передачи собственного объекта Comparator, при третьем вызове конструктора PriorityQueue<Integer> и втором — PriorityQueue<String> используется Comparator с обратной сортировкой, полученный вызовом Collections.reverseOrder().

В последней части добавляется HashSet для уничтожения дубликатов Character. Integer, String и Character изначально работают с PriorityQueue, потому что они обладают «встроенным» естественным упорядочением. При использовании собственного класса с PriorityQueue, следует включить дополнительную реализацию естественного упорядочения или предоставить собственный объект Comparator.

Ключевые методы в Map, Set, List

Ключевые методы	List	Set	Map	Описание
boolean add(element) **boolean** add(index, element)	+ +	+		Добавление элемента.
boolean contains(object) **boolean** containsKey(key) **boolean** containsValue(value)	+	+	 + +	Поиск элемента в коллекции (ключа или значения в карте).
object get(index) object get(key)	+		 +	Получение объекта из коллекции по индексу или ключу.
int indexOf(object)	+			Получение расположения объекта в списке.
Iterator iterator()	+	+		Получение итератора.
Set keySet()			+	Получение множества ключей карты.
put(key, value)			+	Добавление пары ключ/значение в коллекцию.
remove(index) remove(object) remove(key)	+ +	 +	 +	Изъятие объекта из коллекции по индексу, объекту элемента или ключу.
int size()	+	+	+	Возвращение числа элементов в коллекции.
Object[] toArray() T[] toArray(T[])	+	+		Возвращение массива содержащего элементы коллекции.

Краткие теоретические сведения

- В массивах объектам назначаются числовые индексы. Массив содержит объекты заранее известного типа, поэтому преобразование типа при выборке объекта не требуется. Массив может быть многомерным и может использоваться для хранения примитивных типов. Тем не менее изменить размер созданного массива невозможно.
- В Collection хранятся отдельные элементы, а в Map — пары ассоциированных элементов. Механизм параметризации позволяет задать тип объектов, хранимых в контейнере, поэтому поместить в контейнер объект неверного типа невозможно, и элементы не нуждаются в преобразовании типа при выборке. И Collection, и Map автоматически изменяются в размерах при добавлении новых элементов. В контейнерах не могут храниться примитивы, но механизм автоматической упаковки автоматически создает объектные «обертки», сохраняемые в контейнере.
- В контейнере List, как и в массиве, объектам назначаются числовые индексы — таким образом, массивы и List являются упорядоченными контейнерами.
- Используйте ArrayList при частом использовании произвольного доступа к элементам или LinkedList при частом выполнении операций вставки и удаления в середине списка.
- Поведение очередей и стеков обеспечивается контейнером LinkedList.
- Контейнер Map связывает с объектом не целочисленный индекс, а другой объект. Контейнеры HashMap оптимизированы для быстрого доступа, а контейнер TreeMap хранит ключи в отсортированном порядке, но уступает по скорости HashMap. В контейнере LinkedHashMap элементы хранятся в порядке вставки, но хеширование обеспечивает быстрый доступ.
- В контейнере Set каждый объект может храниться только в одном экземпляре. Контейнер HashSet обеспечивает максимальную скорость поиска, а в TreeSet элементы хранятся в отсортированном порядке. В контейнере LinkedHashSet элементы хранятся в порядке вставки.
- Использовать старые классы Vector, Hashtable и Stack в новом коде не нужно.

Задание

Задание А

При выполнении задания следует использовать коллекции и итераторы. Если в задании не указана структура данных, предлагается выбрать её самостоятельно.

Вариант	Задание
1	Осуществляется ввод чисел. В зависимости от знака числа, оно заносится в одну из двух очередей. Выводится очередь с наибольшим количество элементов.
2	Осуществляется ввод слов или строк, символы которых добавляются в коллекцию. Вывести список символов без повторений.
3	Пользователь вводит строки в формате «Ключ=Значение». Ключ — строка, значение — целое число. Вывести список ключей, в алфавитном порядке, определив для каждого максимальное значение.
4	Вводятся вещественные числа. Отсортировать их по убыванию дробной части.
5	Пользователь вводит информацию о прямоугольных треугольниках (длинны катетов). Фигура добавляется в коллекцию в том случае, её площадь отлична от площадей имеющихся в коллекции фигур. Вывести
6	Пользователь вводит строки в формате «Ключ=Значение». Ключ — целое число, значение — целое число. Вывести список ключей, упорядоченных по возрастанию, определив для каждого сумму значений.
7	Вводится последовательность целых положительных чисел. Отсортировать полученные значения по убыванию количества нулевых бит.
8	Пользователь осуществляет ввод целых положительных чисел. Число не может быть добавлено в коллекцию, если в ней уже присутствует число с таким же количеством бит установленных в единицу.
9	Пользователь вводит строки в формате «Ключ=Значение». Ключ и значения являются строками. Вывести список ключей, упорядоченных по убыванию длины ключа, вычислив для каждого ключа значение с максимальной длинной.
10	Осуществляется ввод строк. Следует продублировать строки длина которых меньше средней длины всех строк.
11	Пользователь осуществляет ввод строк, которые добавляются в коллекцию. Строка не может быть добавлена в коллекцию если в ней уже присутствует строка такого же размера.
12	Пользователь вводит строки в формате «Ключ=Значение». Ключ — целое число, значение — целое число. Вывести список ключей, упорядоченных по убыванию, вычислив для каждого сумму дробных частей значений.
13	Осуществляется ввод слов, если пользователь вводит строку, то каждое слово строки добавляется в коллекцию. Следует определить наличие введённого слова в коллекции.

14	Пользователь вводит вещественные числа. Элемент добавляется в коллекцию лишь в том случае если в коллекции не содержится элемента с такой же целой частью.
15	Пользователь вводит строки в формате «Ключ=Значение». Ключ — целое число, значение — строка. Вывести список ключей, упорядоченных по убыванию, определив для каждого строку минимальной длины.

Задание В

В рамках данного задания необходимо переделать задание первой лабораторной работы. Пользователь выбирает поле и направление сортировки, осуществляет ввод данных, по которым создаются элементы класса, которые заносятся в коллекцию. Отсортированная коллекция выводится на экран. Так же пользователю должна быть предоставлена возможность осуществлять поиск по коллекции, который должен быть реализован с использование итераторов.

Вопросы:

Порядок защиты лабораторной работы:

1. Демонстрация программы реализующей выданный вариант лабораторной работы.
2. Выполнение дополнительного задания.
3. Ответ на вопросы по теме текущей лабораторной работы.
4. Ответ на вопросы по предыдущим темам.
5. Предоставление отчёта о выполненной лабораторной работе.