Коллекции

Для динамических структур данных, в Java используются контейнерные классы. Обычно они называются коллекциями.

Java Collections Framework состоит из следующих интерфейсов:

- **List** – это упорядоченный список объектов (иногда еще называют последовательностью объектов). Элементы списка могут вставляться или извлекаться по их индексу в списке (индексация начинается с 0). В эту группу входят: ArrayList, LinkedList;

- **Queue** – это упорядоченный список объектов, объекты расположены в порядке нужном, для их обработки (очередь, стэк).

- **Set** – это неупорядоченная коллекция. Главная особенность множеств – уникальность элементов, то есть один и тот же элемент не может содержаться в множестве дважды. Есть такие имплементации Set интерфейса: HashSet, TreeSet, LinkedHashSet, EnumSet. HashSet хранит элементы в хэш-таблице, что предполагает эффективную реализацию, но не гарантирует порядок итерации элементов. TreeSet хранит элементы в красно-чёрном дереве и упорядочивает элементы по их значениям, эта коллекция существенно медленнее, чем HashSet. LinkedHashSet реализована как хэш-таблица и хранит элементы в связанном списке, в порядке, котором они были вставлены;

- **Map** (не является наследником интерфейса Collection) — отображает ключи в значения и не может иметь дубликаты ключей. Есть три основных имплементации Map интерфейса: **HashMap**, **TreeMap** и **LinkedHashMap**. *HashMap* не гарантирует воспроизводимость порядка вставки элементов. *TreeMap* хранит элементы в красно-чёрном дереве и упорядочивает элементы по их ключам и медленнее, чем *HashMap*.*LinkedHashMap* упорядочивает элементы в порядке их вставки.

Коллекции типа List

ArrayList

Для создания массивов изменяющейся длины используют класс **ArrayList**. Конструктор данного класса может быть без параметров, в этом случае создается массив без элементов. Он может иметь на входе другой массив (коллекцию), тогда он заполняется элементами из переданной ему коллекции. Третий вариант конструктора имеет один входной параметр – число, задающее под сколько элементов, должна быть выделена память (сами элементы при этом не создаются).

Пример:

```
List arrayList = new ArrayList();
```

Следует обратить внимание, что контейнерные классы не являются строго типизированными. Как видно из примера, тип элементов, хранящихся в коллекции, никак не задается. Но тем не менее, некоторые ограничения существуют: контейнерные классы не могут хранить объекты примитивных типов. Поэтому для их хранения следует пользоваться *типами-обертками*.

Для работы с **ArrayList** можно использовать следующие методы:

- void **add**(int index, Object element) — добавляет в массив элемент на заданную позицию;

- boolean **add**(Object o) — добавляет элемент в конец списка;
- boolean **addAll**(Collection c) — добавляет элементы из указанной коллекции к концу массива;
- boolean **addAll**(int **index**, Collection c) — вставляет элементы из указанной коллекции в точку массива с указанным номером;
- void **clear**() — удаляет все элементы из массива;
- boolean **contains**(Object elem) — возвращает истину, если список содержит такой же элемент;
- Object **get**(int index) — возвращает ссылку на элемент с указанным номером;
- int **indexOf**(Object elem) — возвращает номер первого элемента, который равен указанному;
- boolean **isEmpty**() — возвращает истину, если массив не содержит элементов;
- int **lastIndexOf**(Object elem) — возвращает последний элемент, равный указанному;
- Object **remove**(int index) — удаляет элемент с заданным номером;
- protected void **removeRange**(int fromIndex, int toIndex) — удаляет набор элементов, начиная с номера «fromIndex» включительно, и до «toIndex» (он не удаляется);
- Object **set**(int index, Object element) — заменяет элемент с указанным номером на переданный;
- int **size**() — возвращает количество элементов в массиве;
- Object[] **toArray**() — возвращает обычный массив, хранящий все элементы в правильном порядке.

LinkedList

LinkedList представляет структуру данных в виде связанного списка. Класс *LinkedList* имеет следующие конструкторы:

- **LinkedList**() — создает пустой список;
- **LinkedList**(Collection<? **extends** E> collection) — создает список, в который добавляет все элементы коллекции collection.

Пример:

```
List linkedList = new LinkedList();
```

Данный класс имеет практически такой же набор методов, как и *ArrayList*. Разница в основном заключается в скорости выполнения разных операций, но имеется также и несколько своих методов:

- Object **getFirst**() — возвращает первый элемент списка;
- Object **getLast**() — возвращает последний элемент списка;
- ListIterator **listIterator**(int index) — возвращает итератор списка, указывающий на элемент с данным номером;

- Object r**emoveFirst**() — удаляет первый элемент списка;
- Object **removeLast**() — удаляет последний элемент списка.

Коллекции типа Set

Для реализации данного типа коллекции чаще всего используется класс **HashSet**. Класс *HashSet* имеет следующие конструкторы:

- **HashSet**() — создает пустой список;
- **HashSet**(Collection<? **extends** E> collection) — создает хеш-таблицу, в которую добавляет все элементы коллекции collection;
- **HashSet**(int **capacity**) — параметр capacity указывает начальную емкость таблицы, которая по умолчанию равна 16;
- **HashSet**(int capacity, float **koef**) — параметр koef или коэффициент заполнения, значение которого должно быть в пределах от 0.0 до 1.0, указывает, насколько должна быть заполнена емкость объектами, прежде чем произойдет ее расширение. Например, коэффициент 0.75 указывает, что при заполнении емкости на 3/4 произойдет ее расширение.

Особености HashSet:

- HashSet не поддерживает порядок своих элементов, а это значит, что элементы будут возвращены в любом порядке;
- HashSet не разрешает хранить дубликаты. Если вы добавите существующий элемент, то старое значение будет переписано;
- HashSet разрешает добавить в колекцию null значение, но только одно. Пример:

```
Set set = new HashSet();
```

Данные коллекции позволяют выполнять очень ограниченное количество операций, в частности — добавлять элемент, проверять, имеется ли в коллекции такой элемент, и удалять элемент;

- **add**(Object o) — добавляет элемент в коллекцию;
- boolean **contains**(Object o) — проверяет, есть ли элемент в коллекции boolean;
- **remove**(Object o) — удаляет элемент из коллекции.

Коллекции типа Queue

Интерфейс java.util.Queue представляет собой структуру данных, предназначенную для вставки элементов в конец очереди и удаления элементов из начала очереди.

Пример:

```
Queue<String> queue = new PriorityQueue<>();
```

Основные методы:

- E **element**() — возвращает, но не удаляет элемент из начала очереди, если очередь пуста, генерирует исключение NoSuchElementException;
- boolean **offer**(E obj) — добавляет элемент obj в конец очереди, если элемент удачно добавлен, возвращает true, иначе — false;
- E **peek**() — возвращает без удаления элемент из начала очереди, если очередь пуста, возвращает значение null;
- E **poll**() — возвращает с удалением элемент из начала очереди, если очередь пуста, возвращает значение null;
- E **remove**() — возвращает с удалением элемент из начала очереди, если очередь пуста, генерирует исключение NoSuchElementException.