Минусы массива

Базовый интерфейс коллекции java.util.Collection, наследуется от Iterable.

Методы:

size() — размер коллекции;

isEmpty() — проверка пустоты;

contains() — проверка принадлежности объекта коллекции;

add() — добавление элементов в коллекцию;

remove() — удаление элемента из коллекции;

clear() — очистка и удаление всех элементов из коллекции.

Методы add() и remove() возвращают булевское значение.

Методы contains(), remove(), а для некоторых типов коллекций еще и метод add() (12) используют в своей работе метод equals() для сравнения переданного объекта с тем, что уже есть в коллекции. Поэтому для тех классов, экземпляры которых вы собираетесь хранить в коллекциях, важно обеспечить адекватную реализацию equals().

6.2.3 Обход коллекции

1) У коллекции есть метод iterator() (3), унаследованный от интерфейса java.util.Iterable и возвращающий экземпляр java.util.Iterator (3).

Позволяет единообразно обходить элементы любой коллекции. Три метода:

hasNext() — это проверка наличия следующего элемента.

next() — это получение того самого следующего элемента.

remove() — удаляющий текущие элементы из коллекции (который последним возвращен из next())

2) Цикл for each (9) позволяет записать то же самое короче.

3) Методом forEach()

Преимущество Итератора

Список

List — проиндексированный список элементов

Методы:

get() получение элемента по индексу;

set() замена элемента по индексу;

add() добавление элемента по индексу;

remove() удаляет элемент по индексу и возвращает его;

indexOf() и lastIndexOf() ищут элемент в данном списке и возвращают его индекс (с начала или конца, -1 если нет);

subList() возвращает часть списка в виде отдельного экземпляра List.

Реализации:

ArrayList<> реализует поверх массива.

LinkedList основан на двусвязном списке.

Преимущество первого — дешевый доступ по индексу, второго — добавление или удаление элементов

Очередь

Queue — структура данных, обеспечивающая порядок «Первый вошел — первый вышел».

Методы:

add() и offer() добавляет элемент в хвост (исключение и false)

remove() и poll() извлекают первый элемент из головы (исключение и null)

element() и peek() подсмотреть без удаления элемент в голове (исключение и null)

Наследник — Deque, добавляет и удаляет элементы с хвоста и головы, реализации:

ArrayDeque<> поверх массива;

LinkedList<>

Множество

Set — коллекция с уникальными элементами, реализации:

HashSet (на основе хеш‑таблицы; метод hashCode(). HashSet использует хеш‑код объекты для определения ячейки хеш‑таблицы, куда нужно положить этот объект. А метод equals() затем используется для сравнения с элементами, уже лежащими в этой ячейке хеш‑таблицы).

Порядок обхода итератором — обусловлено хеш‑кодом элементов

LinkedHashSet элементы в порядке их добавления.

Наследник: SortedSet (упорядоченное множество) — обеспечивает обход элементов итератором в порядке возрастания.

headSet() (9) возвращает подмножество элементов, меньших чем преданный параметр;

tailSet и subSet — наименьший и наибольший элементы.

Реализация: TreeSet (внутри него самобалансирующийся двоичное дерево поиска, а именно красно‑черное дерево)

Отображение

Не наследует интерфейс Collection, но считается коллекцией.

Индексирует элементы произвольными объектами; методы:

size() размер, количество пар ключ‑значение;

isEmpty() проверка пустоты;

containsKey() и containsValue() есть ли такой ключ или значение;

get() получение значение по ключу;

put() сохранение значение по ключу;

remove() удаляет пару ключ‑значение с заданным ключом;

clear() очищает Map;

Обход (т.к. не коллекция):

keySet() для ключей (вернет Set (17: Set<K>), то есть множество ключей без дубликатов);

values() для значений;

entrySet() ключ и значение;

метод forEach.

Реализация: HashMap, LinkedHashMap, TreeMap (аналогично Set).

6.2.8 Преобразование в массив и обратно

|  |  |
| --- | --- |
| 1.  2.  3. | Collections.shuffle(list);  Collections.sort(list); |

В работе с коллекциями иногда помогает утилитный класс java.util.Collections, в котором есть методы, например, для перетасовывания списка (1) или для сортировки списка (3) и многие другие.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7. | Set<String> set =  Collections.unmodifiableSet(  originalSet);  set.remove("abc");  // throws java.lang.  UnsupportedOperationException |

В классе java.util.Collections есть также семейство методов с именами, начинающимися на unmodifiable. В частности, unmodifiableSet() (2), unmodifiableList(), unmodifiableMap() и так далее. Эти методы возвращают объект‑обертку (1: Set<String>), реализующую тот же интерфейс, что и переданный параметр (3: originalSet), и предоставляющий доступ на чтение к элементам исходной коллекции. Но при этом любая попытка изменить содержимое коллекции (5) через этот объект‑обертку приведет к выбросу (6). То есть мы можем при передаче коллекции в какой‑то метод защититься от его случайного там изменения.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6. | List<Integer> list = ...;  Object[] array1 = list.toArray();  Integer[] array2 =list.toArray(  new Ineger[list.size()]); |

Периодически приходится конвертировать коллекции в классический массив. Для этого в интерфейсе Collections есть метод toArray() (3), который возвращает содержимое коллекции в виде массивов Object[] (3). Это связано с ограничением Generic в Java, о котором мы с вами подробно говорили. Внутри параметризованного класса нельзя создавать экземпляры или массивы типа (1: Integer), указанного в качестве параметра Generic. Если нам хочется получить массив того типа (5: Integer), элементы которого действительно лежат в коллекции, то надо воспользоваться другим перегруженным вариантом метода toArray() (5), который принимает этот массив правильного типа (6: new Ineger[list.size()]). Если массив, подходящего размера, то метод toArray() сложит все элементы в этот самый переданный массив и его же и вернет. Если переданный массив не того размера, которого нужно, то метод toArray() внутри себя перевыделит массив правильного размера и вернет уже новый массив.

|  |  |
| --- | --- |
| 1.  2.  3.  4.  5.  6.  7. | String[] array = {"A", "B", "C"};  Set<String> set1 =  new HashSet<>(Arrays.asList(array));  Set<String> set2 = new HashSet<>();  Collections.addAll(set2, array); |

Выполнить обратную конвертацию из массива в коллекцию тоже не трудно. Можно, например, превратить массив в список при помощи метода Arrays.asList() (4), а потом передать результат в конструктор любого другого класса коллекции (4: new HashSet<>()). Или можно пойти другим путем и создать сначала пустую коллекцию (6), а потом при помощи утилитного метода Collections.addAll() (7) добавить туда все элементы из массива (7: array).