Карты (Map)

Для реализации ассоциативного массива используется класс **HashMap**. Данный класс имеет следующие конструкторы:

- конструктор без параметров создает пустой массив;
- конструктор с одним целочисленным параметром создает пустой массив выделяя при этом память под указанное количество элементов;
- конструктор с одним входным параметром в виде HashMap заносит в массив элементы из переданного массива.

Пример:

```
Map map = new HashMap();
```

Класс имеет следующие методы:

- void **clear**() — удаление всех элементов из массива;
- Object **clone**() — возвращает копию данного массива;
- boolean **containsKey**(Object key) — возвращает истину, если массив содержит элемент с указанным ключом;
- boolean **containsValue**(Object value) — возвращает истину, если массив содержит один или несколько элементов равных указанному;
- Set **entrySet**() — возвращает коллекцию из элементов, содержащихся в массиве;
- Object **get**(Object key) — возвращает объект с указанным ключом, если объекта с таким ключом нет, то возвращается null;
- boolean **isEmpty**() — возвращает истину, если массив пустой;
- Set **keySet**() — возвращает коллекцию, содержащую ключи из данного массива;
- Object **put**(Object key, Object value) — заносит в массив элемент с указанным ключом;
- void **putAll**(Map m) — заносит в массив все элементы из указанного массива, если в массиве уже имеются элементы с совпадающими ключами, они будут заменены новыми;
- Object **remove**(Object key) — даляет объект с указанным ключом;
- int **size**() — возвращает количество элементов в массиве.

Использование дженериков в коллекциях

Если коллекция должна хранить объекты определенного класса и его потомков, переменную коллекции следует создавать следующим образом:

```
List<Point> points = new ArrayList<>();
```

Таким образом, мы получаем переменную коллекции, в которой хранятся точки. И компилятор не позволит занести туда что-то кроме *Point* и его потомков. Кроме того, при получении данных с помощью *get* будет получена ссылка правильного типа.

```
Point p = points.get(3);
```

А в первоначальном виде мы должны были бы писать:

```
Point p = (Point) points.get(3);
```

Причем в этом случае есть шансы получить ошибку из-за несоответствия типа хранимого объекта.

Если вы используете коллекцию типа *Map* в угловых скобках, надо указывать два типа — тип ключа и тип значения.

```
Map<String, Point> pointmap = new HashMap<>();
```

В данном случае мы задаем, что здесь ключом будут строковые объекты, а значениями объекты класса *Point*.

Использование циклов в коллекциях

Допустим у нас есть коллекция:

```
Set<String> stringSet = new HashSet<>() {
        add("1");
        add("2");
        add("3");
        add("4");
};
```

Код, чтобы ее перебрать и вывести на экран с помощью улучшенного цикла for будет выглядеть так:

```
for (String element: stringSet) {
        System.out.println(element);
}
```

Итераторы

Итератор представляет собой интерфейс, позволяющий перебирать содержимое коллекции элемент за элементом. Так как все коллекции, кроме *Map*, наследуют и имплементируют метод **iterator**() от интерфейса *Iterable*, то они имеют возможность итерироваться. Метод *iterator()* создаёт и возвращает ссылку на объект, который реализует интерфейс *Iterator* и позволяет итерировать данную коллекцию. Если имеется коллекция *Points*, то создать итератор можно следующим образом:

```
Iterator iterator = points.iterator();
```

Затем можно перебирать все элементы коллекции с помощью методов итератора.

- boolean **hasNext**() — возвращает истину, если у итератора еще остались элементы для перебора;
- Object **next**() — переходит на следующий объект и возвращает пройденный объект;
- void **remove**() — удаляет из коллекции элемент, который только что был пройден.

Пример перебора коллекции с помощью итератора:

```
while(iterator.hasNext()){
        ((Point) iterator.next()).print();
}
```

Как и в самих коллекциях, итератор может использовать угловые скобки с указанием типа.

```
Iterator<Point> iterator = points.iterator();
while(iterator.hasNext()){
        iterator.next().print();
}
```

В данной ситуации мы получили некоторое упрощение записи, так как метод next итератора сразу возвращает объекты класса *Point* и нам не потребовалось использовать приведение типов.

Итераторы существуют только для коллекции *Set* или *List*. Коллекции типа *Map* напрямую перебираться итераторами не могут. Можно извлечь набор ключей в виде *Set* и перебирая его итератором перебрать *Map*. Например:

```
Map<String, Integer> m = new HashMap<String, Integer>();
...
Set<String> keys = m.keySet();
Iterator<String> iterator = keys.iterator();
while (iterator.hasNext()) {
        String s = iterator.next();
        System.out.println(s + " " + m.get(s));
}
```

Следует помнить, что для итераторов не существует понятия “текущий объект”, итератор указывает не на объект, а между объектами.

Данный вариант итератора будет работать практически с любыми коллекциями. Для коллекций типа «список», таких как *ArrayList* или *LinkedList* существует специальный итератор **ListIterator**. Для его получения данные коллекции используют метод **listIterator**. Данный вариант итератора обладает значительно более широкими возможностями, соответственно, большим количеством методов:

- void **add**(Object o) — вставляет элемент в список;

- boolean **hasPrevious**() — возвращает истину, если еще есть элементы для перебора, при прохождении в направлении начала списка;
- int **nextIndex**() — возвращает номер следующего элемента;
- Object **previous**() — сдвигается на один элемент в направлении начала списка и возвращает пройденный объект;
- int **previousIndex**() — возвращает номер предыдущего элемента;
- void **remove**() — удаляет элемент который был пройден последним;
- void **set**(Object o) — присваивает новое значение элементу, который был пройден последним.