Язык Kotlin: коллекции

Массивы удобны для хранения данных, однако имеют одно существенное ограничение: количество элементов в них задаётся при создании. Нельзя добавить новый элемент или удалить существующий. Да и индексом в массиве может быть только целое число начиная с 0. Поэтому помимо массивов существуют и другие формы хранения данных, в целом называемые *коллекциями*.

В Kotlin есть списки (`List`), словари (`Map`) и множества (`Set`). Каждая из таких коллекций имеет свои уникальные особенности. И каждая коллекция может быть изменяемой (mutable), и тогда с ней можно совершать операции добавления или удаления элементов, или неизменяемой – в этом случае набор элементов задаётся при создании и изменить его уже нельзя.

Списки

Список – это упорядоченная коллекция элементов. Список может быть изменяемым (`MutableList`) или неизменяемым (`List`).

Для создания элементов используется функция `listOf` или `mutableListOf` – в зависимости от того, какой тип списка требуется создать. В этом примере создаётся пустой список целых чисел, и потом в него добавляются значения:

```
val list: MutableList<Int> = mutableListOf()
list.add(3)
list.add(5)
list.add(8)
```

Перебирать значения списка можно так же, как и массивы:

```
for (n in list)
    print("${n} ") // 3 5 8
```

Чтобы добавить элемент не в конец списка, а в определенную позицию, нужно в методе `add` указать сначала желаемую позицию (начиная с 0), а затем сам добавляемый элемент:

```
list.add(3)    // 0-й элемент
list.add(5)    // 1-й элемент
list.add(8)    // 2-й элемент
list.add(1, 7) // Вставить число 7 как 1-й элемент (остальные сместятся)
for (n in list)
    print("${n} ") // 3 7 5 8
```

Для удаления элемента из какой-либо позиции служит метод `removeAt`:

```
list.add(3)      // 0-й элемент
list.add(5)      // 1-й элемент
list.add(8)      // 2-й элемент
list.removeAt(1) // Удалить 1-й элемент
for (n in list)
    print("${n} ") // 3 8
```

Можно удалить также и сам элемент по значению, а не по индексу:

```
list.add(3)
list.add(5)
list.add(8)
list.remove(3) // Удалить элемент со значением “3”
for (n in list)
    print("${n} ") // 5 8
```

Наконец, метод `clear` очищает весь список.

Чтобы проверить есть ли определенный элемент в списке, используется функция `contains`:

```
list.add(3)
list.add(5)
list.add(8)
println(list.contains(1)) // false
println(list.contains(5)) // true
```

Если требуется не просто проверить наличие элемента, но и получить его индекс в коллекции, можно использовать функции `indexOf` и `indexOfFirst`. Разница между ними в том, что `indexOf` в качестве параметра принимает сам элемент, который нужно найти – обычно это число или строка. Функция же `indexOfFirst` использует программный код для проверки подходит ли элемент или нет: каждый элемент списка передаётся в виде переменной `it`, и код должен вернуть `true` если элемент подходит, и `false` если нет. Как только код возвращает `true`, функция `indexOfFirst` прекращает работу и возвращает индекс данного элемента:

```
list.add(3)
list.add(5)
list.add(8)
println(list.indexOf(5)) // 1
println(list.indexOfFirst(it == 8)) // 2
```

Аналогичным образом работает и функция `find`, только она возвращает не индекс элемента, а сам элемент.

Словари

Словарь – это особая коллекция, в которой хранятся пары вида «ключ : значение», при этом ключ должен быть уникальным. В Kotlin словари представлены классом `Map`.

В каком-то смысле массив тоже можно назвать словарём: в нём ключами являются последовательные числа от 0 до номера последнего элемента массива. Но в настоящем словаре ключом могут быть не только числа, но и, например, строки.

Чтобы создать массив, используется функция `mapOf`:

```
var phoneCodes: MutableMap<String, String> = mutableMapOf(
    "+7" to "Россия",
    "+30" to "Греция",
    "+46" to "Швеция",
    "+44" to "Великобритания")
```

В этом примере создаётся словарь, в котором и ключи и значения являются строками. Чтобы получить значение, связанное с ключом, нужно просто использовать квадратные скобки:

```
print(phoneCodes["+30"]) // Греция
```

Если в словаре нет пары с указанным ключом, то будет возвращено значение `null`:

```
print(phoneCodes["+50"]) // null
```

Для добавления нового значения используется функция `put`:

```
phoneCodes.put("+86", "Китай")
```

А если нужно исправить какое-то значение, то можно просто присвоить ему новые данные:

```
phoneCodes["+30"] = "Греция (но это не точно)"
print(phoneCodes["+30"]) // Греция (но это не точно)
```

Удаление элемента, как и в случае списка, производится функцией remove:

```
phoneCodes.remove("+46")
println(phoneCodes["+46"]) // null, элемента больше нет
```

Для получения списка всех ключей используется свойство `keys`:

```
for (k in phoneCodes.keys)
    println(k) // +7 +30 +46 +44
```

Наконец, можно сделать перебор всех элементов словаря:

```
for (code in phoneCodes)
    println("Код ${code.key} - это страна ${code.value}")
```

Этот код выведет на экран список всех кодов и стран:

```
Код +7 - это страна Россия
Код +30 - это страна Греция
Код +46 - это страна Швеция
Код +44 - это страна Великобритания
```

Множества

Множество – это ещё одна особая коллекция языка Kotlin. Множество содержит какие-либо значения, каждое из которых может входить в множество только в единственном экземпляре. Например, следующий пример создаёт множество фруктов:

```
var fruits = mutableSetOf("Яблоко", "Апельсин", "Груша")
for (fruit in fruits)
    println(fruit) // Яблоко Апельсин Груша
```

Теперь можно добавить новые элементы в множество:

```
fruits.add("Арбуз")
fruits.add("Яблоко")
for (fruit in fruits)
    println(fruit) // Яблоко Апельсин Груша Арбуз
```

Арбуз добавился в множество фруктов, а яблоко – нет, потому что в множестве уже было яблоко. Чтобы проверить есть ли в множестве определенный элемент, используется функция `contains`:

```
println(fruits.contains("Апельсин")) // true
println(fruits.contains("Манго"))    // false... а жаль!
```

Чтобы удалить элемент, используется функция minus:

```
fruits.minus("Груша")
for (fruit in fruits)
    println(fruit) // Яблоко Апельсин Арбуз
```

Задание

Напишите программу, которая имитирует работу телефонной станции. В программе должно быть два класса:

- `Abonent` – содержит поля с именем абонента и его номером, а также список входящих и исходящих звонков
- `Station` – содержит список абонентов и два метода:
    - `call(from: String, to: String)` – ищет абонентов с именами `from` и `to`, и если они найдены, то первому добавляет в журнал вызовов строку «`Исходящий к <…>`», где вместо `<…>` подставляется имя абонента `to`, а второму добавляет в журнал вызовов строку «`Входящий от <…>`», где вместо `<…>` подставляется имя абонента `from`.
    - `showStat()` – выводит полные журналы звонков каждого абонента в формате:

      ```
      Журнал звонков абонента <…>:
          Входящий от <…>
          Входящий от <…>
          Исходящий от <…>

      Журнал звонков абонента <…>:
          Исходящий от <…>
          Исходящий от <…>
      ```

В функции `main` следует создать объект `Station`, добавить в него несколько абонентов, несколько раз вызвать функцию `call()` для имитации звонков между разными абонентами, а затем вызвать функцию `showStat()` для отображения журнала звонков.