Коллекции. Collection

Протокол Collection наследуется от Sequence, добавляя индексы, сабскрипты и обязательство не разрушать данные при проходе итератора. Благодаря последнему можно использовать этот протокол чтобы показать, что ваша последовательность стабильная даже если вам не нужна остальная функциональность коллекций.

При создании собственной коллекции обязательными для имплементации являются startIndex, endIndex, методы subscript(position:) и index(after:). Для остальных уже есть реализация по умолчанию. К сожалению нет никакого механизма помогающего это понять. Приходится играть в угадайку с компилятором либо искать ответы в документации к протоколу.

Рассмотрим простейшую коллекцию - стэк. В данной реализации он может хранить в себе строки и выдавать их по принципу последний вошел - первый вышел. Для этого у него есть методы push() и pop(). Хранит он данные в массиве, но это может быть любой другой контейнер, сохраняющий порядок элементов.

```
// --- Коллекции. Collection CODE SNIPPET #1 --- //

struct Stack {
    private var privateStorage: Array<String> = []

    mutating func push(_ element: String) {
        privateStorage.append(element)
    }

    mutating func pop() -> String? {
        return privateStorage.popLast()
    }
}

var stack = Stack()

stack.push("Hello")
stack.push("World")
stack.push("?")
stack.pop()
stack.push("!")
```

Добавим ему поддержку протокола Collection через расширение. Для этого реализуем минимальный необходимый набор для поддержки протокола. assossiatedtype указывать явно не обязательно. Swift сам поймет, что Element - это String, а Index - это Int.

// --- Коллекции. Collection CODE SNIPPET #2 --- //

```
extension Stack: Collection {
    var startIndex: Int {
        return 0
    }

    var endIndex: Int {
        return privateStorage.count
    }

    func index(after i: Int) -> Int {
        return i + 1
    }

    subscript(position: Int) -> String {
        return privateStorage[position]
    }
}
```

Помимо них в протоколе имеются и другие assosiatedtype. Iterator, унаследованный от Sequence, будет иметь тип IndexingIterator<Stack>. Это простой итератор который оборачивает коллекцию и использует ее для получения следующего элемента. SubSequence также полученный от Sequence будет иметь тип Slice<Stack>. Имеет смысл заменить его на тип самой коллекции. Чуть позже я покажу как это сделать. IndexDistance по умолчанию имеет тип Int и используется для указания разницы между двумя индексами. Его менять не нужно. Indices - DefaultIndices<Int>. Еще один врапер над коллекцией. Он содержит все валидные индексы коллекции.

Поддержав этот протокол мы получаем бесплатно кучу полезных методов. Можно использовать коллекцию в цикле for in, можно узнать количество элементов в коллекции, использовать в сабскриптах диапазоны, передавать ее в функции в качестве Sequence, сортировать и трансформировать коллекцию. На слайде несколько простых примеров, а трансформации будут рассмотрены в других лекциях.

// --- Коллекции. Collection CODE SNIPPET #3 --- //

```
for element in stack {
    print("\(element) ")
}

stack.count // 3
stack.isEmpty // false

Array(stack[1...2]) // ["World", "!"]
```

Также можно добавить поддержку ExpressibleByArrayLiteral. Это позволит инициализировать наш стэк с помощью литералов массивов. Все что нам нужно - это добавить один конструктор. Он получает список элементов и в цикле добавляет их в стэк.

```
// --- Коллекции. Collection CODE SNIPPET #4 --- //

extension Stack: ExpressibleByArrayLiteral {
    public init(arrayLiteral elements: String...) {
        for element in elements {
            push(element)
        }
    }
}

let stackFromArrayLiteral: Stack = ["one", "two", "three"]
```

Еще одним важным понятием которое мы сегодня рассмотрим являются индексы. В предыдущих примерах в качестве индекса мы для удобства использовали Int. Но коллекция может иметь индекс любого сравнимого(Comparable) типа. Именно поэтому не стоит использовать индексы вручную или пытаться найти следующий прибавив единицу. Один из таких примеров это String. Ее индексом является структура String.Index. Для работы с индексами нужно использовать специальные методы коллекции. Рассмотрим некоторые из них.

```
// --- Коллекции. Collection CODE SNIPPET #5 --- //

let string = "Simple string"

var indexInString1 = string.index(of: "s")!
var indexInString2 = indexInString1
indexInString2 = string.index(after: indexInString2)
string.formIndex(after: &indexInString2)

var indexDistance = string.distance(from: indexInString1,
                                    to: indexInString2)

indexInString1 = string.index(indexInString1, offsetBy: indexDistance)

indexInString1 == indexInString2 // true
```

Еще в коллекции есть два специальных индекса: startIndex и endIndex. Они указывают на первый элемент и на элемент следующий за последним соответственно. Т.е. endIndex не указывает на валидный элемент в коллекции. Это позволяет писать циклы с условиями while index < endIndex.