Swift Generics

Problem

```
1  func swapTwoInts(_ a: inout Int, _ b: inout Int) {
2      let temporaryA = a
3      a = b
4      b = temporaryA
5  }
```

Problem

```swift
func swapTwoInts(_ a: inout Int, _ b: inout Int) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

```swift
var someInt = 3
var anotherInt = 107
swapTwoInts(&someInt, &anotherInt)
print("someInt is now \(someInt), and anotherInt is now \(anotherInt)")
// Prints "someInt is now 107, and anotherInt is now 3"
```

Problem

```swift
func swapTwoStrings(_ a: inout String, _ b: inout String) {
    let temporaryA = a
    a = b
    b = temporaryA
}

func swapTwoDoubles(_ a: inout Double, _ b: inout Double) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

Решение

Generic Functions

```swift
func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

Generic Functions

```swift
func swapTwoValues<T>(_ a: inout T, _ b: inout T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

```swift
func swapTwoInts(_ a: inout Int, _ b: inout Int)
func swapTwoValues<T>(_ a: inout T, _ b: inout T)
```

Generic Functions

```
1  var someInt = 3
2  var anotherInt = 107
3  swapTwoValues(&someInt, &anotherInt)
4  // someInt is now 107, and anotherInt is now 3
5
6  var someString = "hello"
7  var anotherString = "world"
8  swapTwoValues(&someString, &anotherString)
9  // someString is now "world", and anotherString is now "hello"
```

Generic Functions

1. `func funcWithMultiplePlaceholders<T, U, V>(_ a: T, _ b: U, _ c: V)`

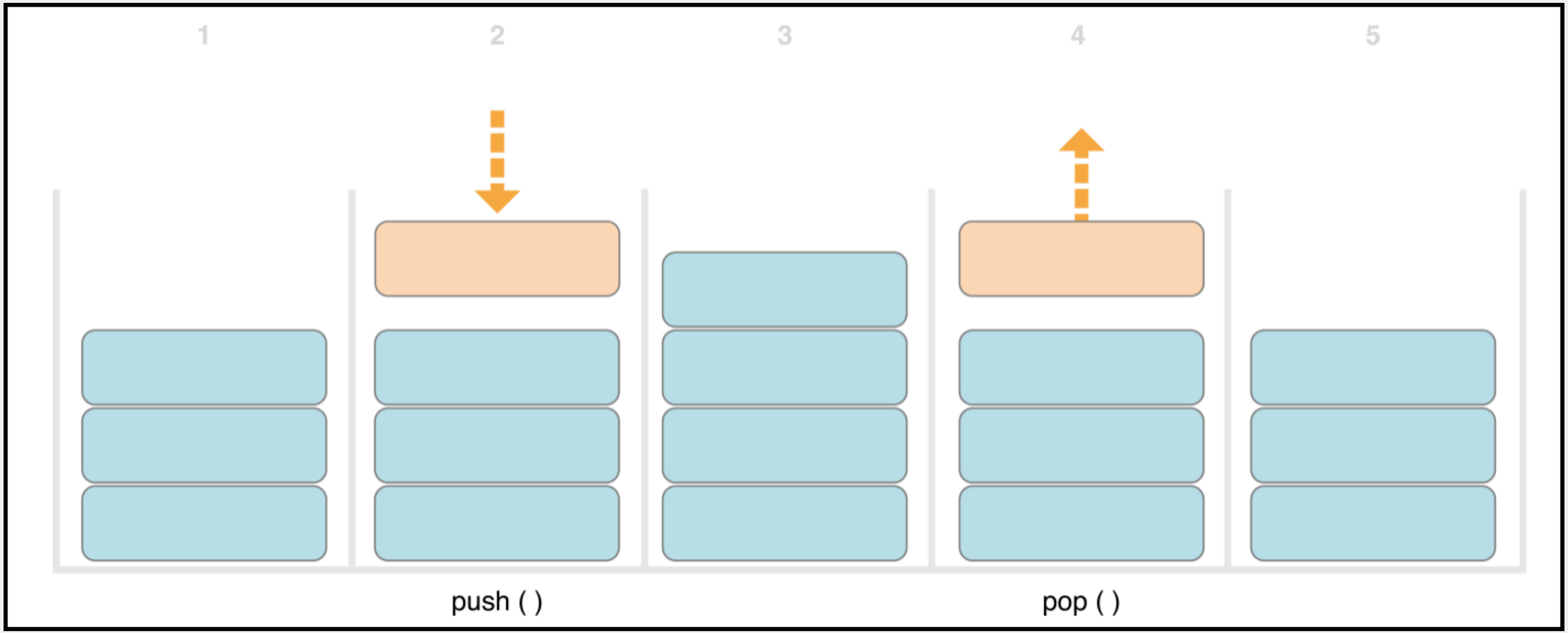
2. `public struct Array<Element>`

Generic vs Any

```
extension Array {
    func reduce<Result>(_ initial: Result, _ combine: (Result, Element) -> Result) -> Result
}
```

```
extension Array {
    func reduce(Any, (Any, Any) -> Any) -> Any
}
```

Generic Types



Generic Types

```
1  struct IntStack {
2      var items = [Int]()
3      mutating func push(_ item: Int) {
4          items.append(item)
5      }
6      mutating func pop() -> Int {
7          return items.removeLast()
8      }
9  }
```

Devteam

Generic Types

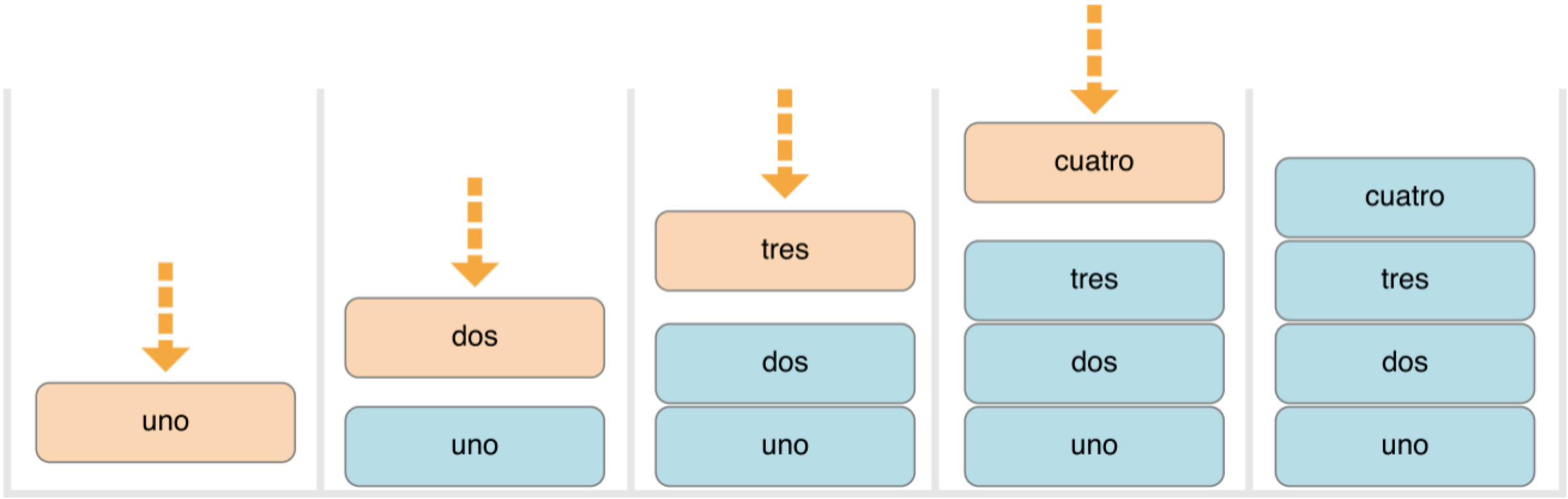
```
1  struct Stack<Element> {
2      var items = [Element]()
3      mutating func push(_ item: Element) {
4          items.append(item)
5      }
6      mutating func pop() -> Element {
7          return items.removeLast()
8      }
9  }
```

Devteam

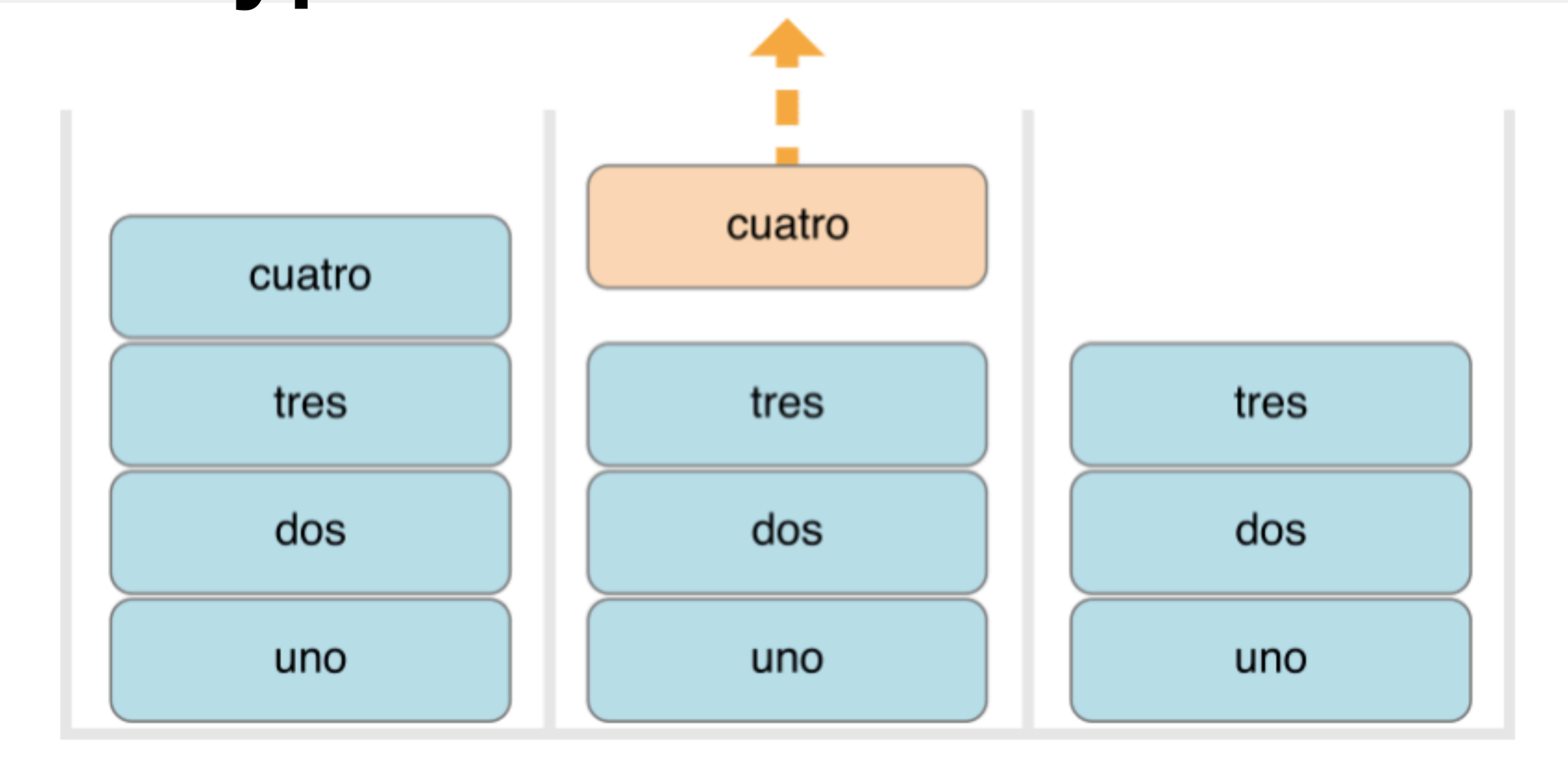
Generic Types

```
1  var stackOfStrings = Stack<String>()
2  stackOfStrings.push("uno")
3  stackOfStrings.push("dos")
4  stackOfStrings.push("tres")
5  stackOfStrings.push("cuatro")
6  // the stack now contains 4 strings
```

Generic Types



Generic Types




```
let fromTheTop = stackOfStrings.pop()
// fromTheTop is equal to "cuatro", and the stack now contains 3 strings
```

Extending a Generic Type

```
1  extension Stack {
2      var topItem: Element? {
3          return items.isEmpty ? nil : items[items.count - 1]
4      }
5  }
```

```
1  if let topItem = stackOfStrings.topItem {
2      print("The top item on the stack is \(topItem).")
3  }
4  // Prints "The top item on the stack is tres."
```

Type Constraints

Generic Structure

Dictionary

A collection whose elements are key-value pairs.

Declaration

```
struct Dictionary<Key, Value> where Key : Hashable
```

Type Constraints

```swift
func someFunction<T: SomeClass, U: SomeProtocol>(someT: T, someU: U) {
    // function body goes here
}
```

Type Constraints

```swift
func findIndex(ofString valueToFind: String, in array: [String]) -> Int? {
    for (index, value) in array.enumerated() {
        if value == valueToFind {
            return index
        }
    }
    return nil
}
```

```swift
let strings = ["cat", "dog", "llama", "parakeet", "terrapin"]
if let foundIndex = findIndex(ofString: "llama", in: strings) {
    print("The index of llama is \(foundIndex)")
}
// Prints "The index of llama is 2"
```

Type Constraints

```
1  func findIndex<T>(of valueToFind: T, in array:[T]) -> Int? {
2      for (index, value) in array.enumerated() {
3          if value == valueToFind {
4              return index
5          }
6      }
7      return nil
8  }
```

Сбербанк Онлайн

Devteam

Type Constraints

```
1  func findIndex<T>(of valueToFind: T, in array:[T]) -> Int? {
2      for (index, value) in array.enumerated() {
3          if value == valueToFind {
4              return index
5          }
6      }
7      return nil
8  }
```

Сбербанк Онлайн

Devteam

Type Constraints

```swift
func findIndex<T: Equatable>(of valueToFind: T, in array:[T]) -> Int? {
    for (index, value) in array.enumerated() {
        if value == valueToFind {
            return index
        }
    }
    return nil
}
```

```swift
let doubleIndex = findIndex(of: 9.3, in: [3.14159, 0.1, 0.25])
// doubleIndex is an optional Int with no value, because 9.3 isn't in the array
let stringIndex = findIndex(of: "Andrea", in: ["Mike", "Malcolm", "Andrea"])
// stringIndex is an optional Int containing a value of 2
```

Devteam

Задача

```
func sum<T>(_ a: T, _ b: T) -> T {

}
```

Сбербанк Онлайн

Devteam

Associated Types

```
1  protocol Container {
2      associatedtype Item
3      mutating func append(_ item: Item)
4      var count: Int { get }
5      subscript(i: Int) -> Item { get }
6  }
```

Associated Types

```
struct IntStack: Container {
    // original IntStack implementation
    var items = [Int]()
    mutating func push(_ item: Int) {
        items.append(item)
    }
    mutating func pop() -> Int {
        return items.removeLast()
    }
    // conformance to the Container protocol
    typealias Item = Int
    mutating func append(_ item: Int) {
        self.push(item)
    }
    var count: Int {
        return items.count
    }
    subscript(i: Int) -> Int {
        return items[i]
    }
}
```

Devteam

Associated Types

```
struct Stack<Element>: Container {
    // original Stack<Element> implementation
    var items = [Element]()
    mutating func push(_ item: Element) {
        items.append(item)
    }
    mutating func pop() -> Element {
        return items.removeLast()
    }
    // conformance to the Container protocol
    mutating func append(_ item: Element) {
        self.push(item)
    }
    var count: Int {
        return items.count
    }
    subscript(i: Int) -> Element {
        return items[i]
    }
}
```

Devteam

Associated Types

Generic Structure

Array

An ordered, random-access collection.

Declaration

```
struct Array<Element>
```

```
extension Array: Container {}
```

Devteam

Associated Types

```
1  protocol Container {
2      associatedtype Item: Equatable
3      mutating func append(_ item: Item)
4      var count: Int { get }
5      subscript(i: Int) -> Item { get }
6  }
```

Сбербанк Онлайн

Devteam

Associated Types

```
1  protocol SuffixableContainer: Container {
2      associatedtype Suffix: SuffixableContainer where Suffix.Item == Item
3      func suffix(_ size: Int) -> Suffix
4  }
```

Сбербанк Онлайн

Devteam

Associated Types

```
extension Stack: SuffixableContainer {
    func suffix(_ size: Int) -> Stack {
        var result = Stack()
        for index in (count-size)..<count {
            result.append(self[index])
        }
        return result
    }
    // Inferred that Suffix is Stack.
}
var stackOfInts = Stack<Int>()
stackOfInts.append(10)
stackOfInts.append(20)
stackOfInts.append(30)
let suffix = stackOfInts.suffix(2)
// suffix contains 20 and 30
```

Devteam

Associated Types

```
1  extension IntStack: SuffixableContainer {
2      func suffix(_ size: Int) -> Stack<Int> {
3          var result = Stack<Int>()
4          for index in (count-size)..<count {
5              result.append(self[index])
6          }
7          return result
8      }
9      // Inferred that Suffix is Stack<Int>.
10 }
```

Devteam

Generic Where Clauses

```swift
func allItemsMatch<C1: Container, C2: Container>
    (_ someContainer: C1, _ anotherContainer: C2) -> Bool
    where C1.Item == C2.Item, C1.Item: Equatable {

        // Check that both containers contain the same number of items.
        if someContainer.count != anotherContainer.count {
            return false
        }

        // Check each pair of items to see if they're equivalent.
        for i in 0..<someContainer.count {
            if someContainer[i] != anotherContainer[i] {
                return false
            }
        }

        // All items match, so return true.
        return true
}
```

Devteam

Generic Where Clauses

```
var stackOfStrings = Stack<String>()
stackOfStrings.push("uno")
stackOfStrings.push("dos")
stackOfStrings.push("tres")

var arrayOfStrings = ["uno", "dos", "tres"]

if allItemsMatch(stackOfStrings, arrayOfStrings) {
    print("All items match.")
} else {
    print("Not all items match.")
}
// Prints "All items match."
```

Extensions with a Generic Where Clause

```
1  extension Stack where Element: Equatable {
2      func isTop(_ item: Element) -> Bool {
3          guard let topItem = items.last else {
4              return false
5          }
6          return topItem == item
7      }
8  }
```

Extensions with a Generic Where Clause

```
1  if stackOfStrings.isTop("tres") {
2      print("Top element is tres.")
3  } else {
4      print("Top element is something else.")
5  }
6  // Prints "Top element is tres."
```

Extensions with a Generic Where Clause

```
extension Container where Item: Equatable {
    func startsWith(_ item: Item) -> Bool {
        return count >= 1 && self[0] == item
    }
}
```

```
if [9, 9, 9].startsWith(42) {
    print("Starts with 42.")
} else {
    print("Starts with something else.")
}
// Prints "Starts with something else."
```

Devteam

Extensions with a Generic Where Clause

```
1  extension Container where Item == Double {
2      func average() -> Double {
3          var sum = 0.0
4          for index in 0..<count {
5              sum += self[index]
6          }
7          return sum / Double(count)
8      }
9  }
10 print([1260.0, 1200.0, 98.6, 37.0].average())
11 // Prints "648.9"
```

Associated Types with a Generic Where Clause

```
1  protocol Container {
2      associatedtype Item
3      mutating func append(_ item: Item)
4      var count: Int { get }
5      subscript(i: Int) -> Item { get }
6
7      associatedtype Iterator: IteratorProtocol where Iterator.Element ==
     Item
8      func makeIterator() -> Iterator
9  }
```

Associated Types with a Generic Where Clause

```
protocol ComparableContainer: Container where Item: Comparable { }
```

Сбербанк Онлайн

Devteam

Задача

```
struct Bucket<Bucketable> {

    func add(item: Bucketable) {}

    func remove() {}
}
```

Сбербанк Онлайн

Devteam

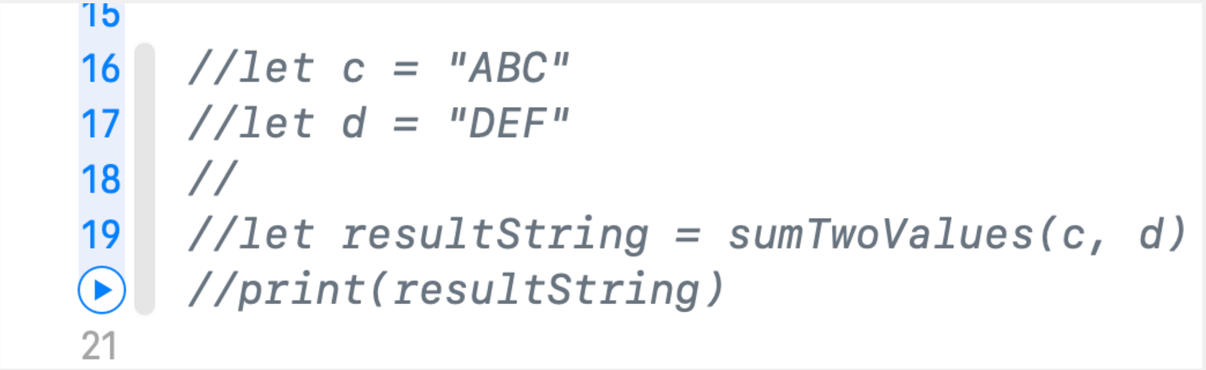
Read

- https://docs.swift.org/swift-book/LanguageGuide/Generics.html
- https://www.swiftbysundell.com/basics/generics/
- https://www.objc.io/blog/2019/03/05/generics-vs-any/
- https://developer.apple.com/videos/play/wwdc2018/406/
- https://youtu.be/RvSoXVxN7Aw

- https://docs.swift.org/swift-book/LanguageGuide/OpaqueTypes.html
- https://www.swiftbysundell.com/articles/opaque-return-types-in-swift/
- https://medium.com/@pavbox/swift-5-1-opaque-types-bc45c0f700ae

Homework

1. В Playground SwiftGenerics_01 раскомментировать указанный код и сделать так, чтобы он работал.



```
15
16  //let c = "ABC"
17  //let d = "DEF"
18  //
19  //let resultString = sumTwoValues(c, d)
    //print(resultString)
21
```

2. В Playground SwiftGenerics_02 создать систему generic-типов. Типы могут быть свои либо можно использовать предложенный ниже вариант

   “Реализовать базовый протокол для контейнеров. Контейнеры должны отвечать, сколько они содержат элементов, добавлять новые элементы и возвращать элемент по индексу. На основе базового протокола реализовать универсальный связанный список и универсальную очередь (FIFO) в виде структуры или класса. ”

3. * В Playground SwiftGenerics_03 реализовать универсальный связанный список на основе enum.

Сбербанк Онлайн

СРОК: к 5 июля

Devteam