
Zimin
SWIFT
MAGIC

- Nested Types
- Generics
- Protocols
- Protocol-Oriented Programming
- Mirror Type and Reflection

Why Swift?

Why Swift?

Стоит упомянуть, что использование структур для всего и везде (толк от Uber(a))

It’s fast 🚀

It’s powerful 💪

It’s awesome 🏆

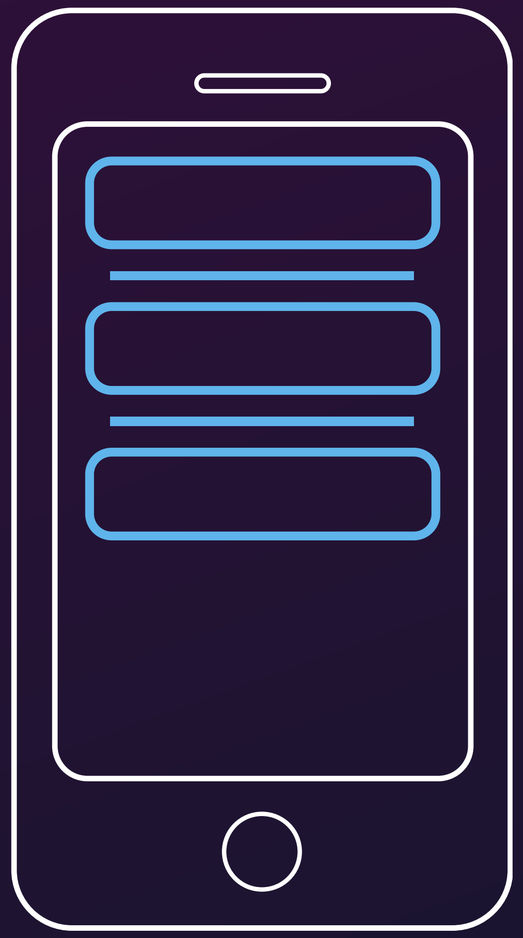
Colors

Colors

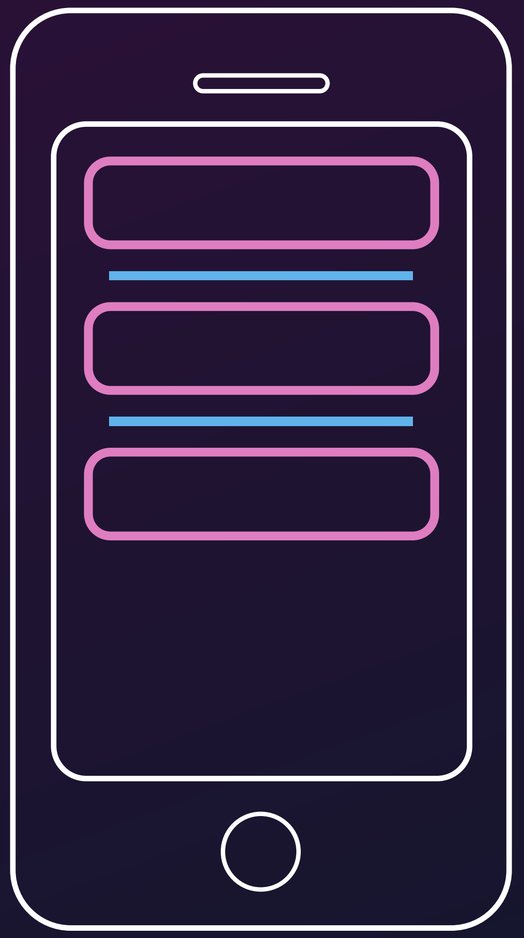
```
view.backgroundColor =
 UIColor(hexString: "6094EA")
```


```
view.backgroundColor = UIColor(.strongGray)
view.backgroundColor = UIColor.strongGray
```

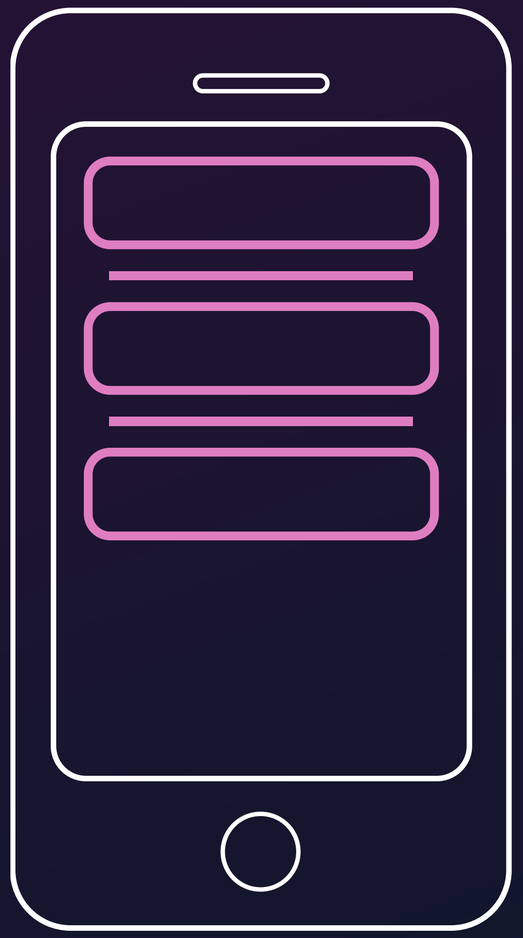
Old



Expectation



Reality



```
extension UIColor {

    static var tableElementsSeparator: UIColor {
        return UIColor(.strongGray)
    }

    static var barsSeparator: UIColor {
        return UIColor(.red)
    }

    static var barsBackground: UIColor {
        return UIColor(.gray)
    }

}
```

Type Omission

```
extension UIColor {

    static var tableElementsSeparator: UIColor {
        return UIColor(.strongGray)
    }

    static var barsSeparator: UIColor {
        return UIColor(.red)
    }

    static var barsBackground: UIColor {
        return UIColor(.gray)
    }

    static var toolBox: UIColor {
        return UIColor(.gray)
    }

    static var activeControl: UIColor {
```

```
extension UIColor {

    static var tableElementsSeparator: UIColor {
        return UIColor(.strongGray)
    }

    static var barsSeparator: UIColor {
        return UIColor(.red)
    }

    static var barsBackground: UIColor {
        return UIColor(.gray)
    }

}
```

```
extension UIColor {

    struct Separator {

        static var tableElements: UIColor {
            return UIColor(.strongGray)
        }

        static var bars: UIColor {
            return UIColor(.red)
        }

    }

}
```

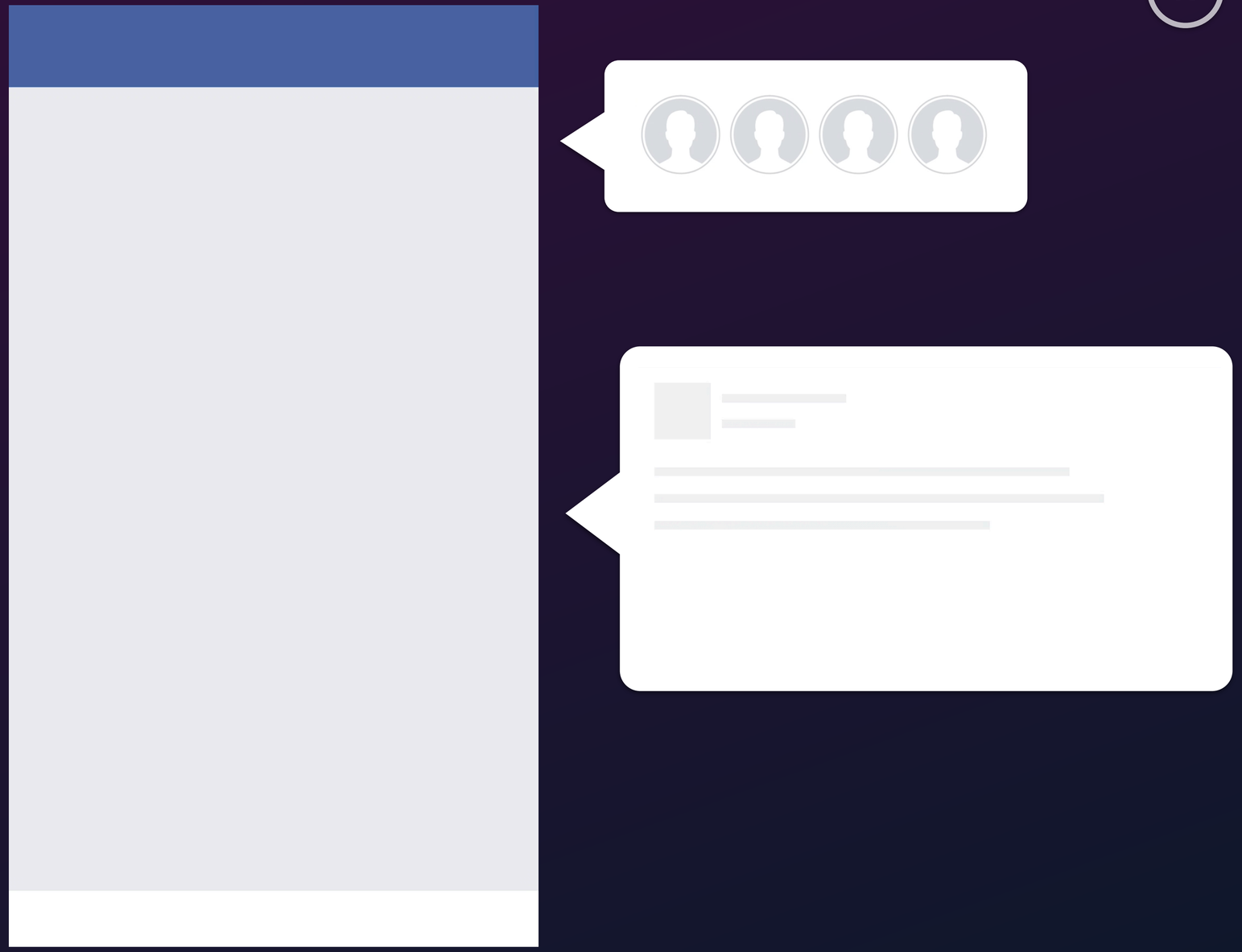
Struct	Enum
Separator. UIColor **bars** Separator init() UIColor tableElements	Separator. UIColor **bars** UIColor tableElements

```
extension UIColor {

    enum Separator {

        static var tableElements: UIColor {
            return UIColor(.strongGray)
        }

        static var bars: UIColor {
            return UIColor(.red)
        }

    }

}
```

```
let activeStateColor =
 UIColor.Control.Button.activeStateTitle

button.setTitleColor(activeStateColor, for: .normal)
```

Generics

Generics<T>



```
struct StoriesDisplayCollection {

  var stories: Results<StoryEntity>

  var count: Int {
    return stories.count
  }

  func nameAtIndex(index: Int) -> String {
    return stories[index].name
  }

}
```

```
struct StoriesDisplayCollection {

  var stories: Results<StoryEntity>

  var count: Int {
    return stories.count
  }

  func nameAtIndex(index: Int) -> String {
    return stories[index].name
  }

}
```

Stories

```
struct StoriesDisplayCollection {

  var stories: Results<StoryEntity>

  var count: Int {
    return stories.count
  }

  func nameAtIndex(index: Int) -> String {
    return stories[index].name
  }

}
```

```
struct StoriesDisplayCollection {

  var stories: Results/List<StoryEntity>

}
```

```
struct StoriesDisplayCollection {

  var stories: TemplateModelsCollection<StoryEntity>

}
```

```
struct TemplateModelsCollection<T: TemplateEntity>
    where T: Object {

}
```

```
enum Content {

  case result(Results<T>)
  case list(List<T>)
  case empty

  subscript(index: Int) -> T {
    switch self {
    case let .result(model):
      return model[index]
    case let .list(list):
      return list[index]
     case .empty:
      return T.templateEntity
    }
  }

}
```

Nested

Same T, Swift 3.1+

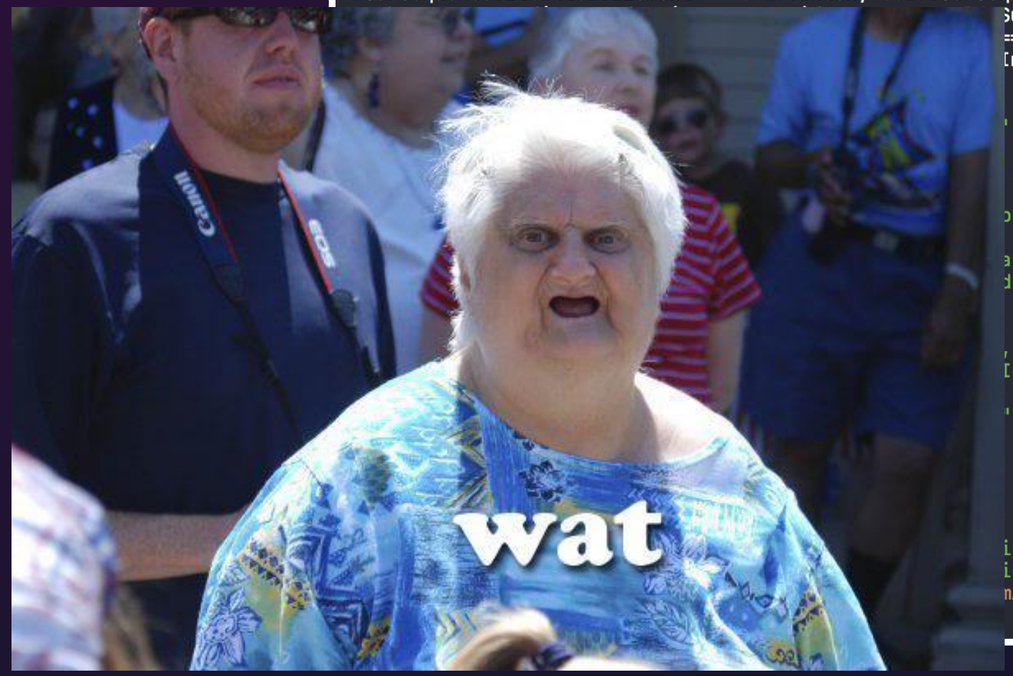
```
struct TemplateModelsCollection<T: TemplateEntity>
    where T: Object {

  init(dataCollection: Results<T>, templatesCount: Int = 5) {
    content = Content.result(dataCollection)
    values = List<T>()
    generateFakeData(templatesCount: templatesCount)
  }

}
```

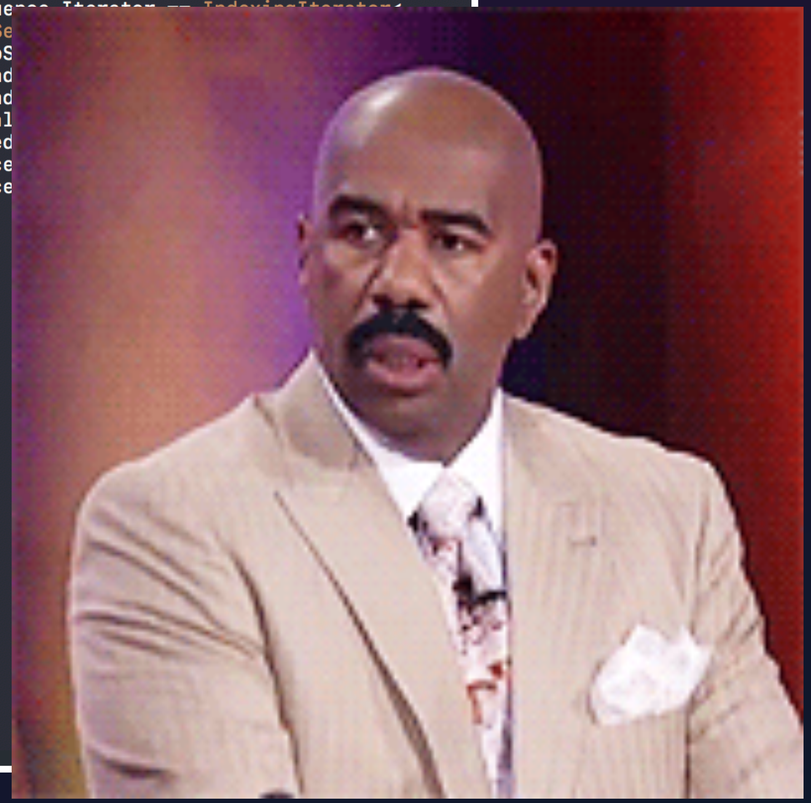
Template One

```
struct TemplateModelsCollection<T: TemplateEntity>
    where T: Object {

  var count: Int {
    if isLoading && content.count == 0 {
      return values.count
    } else {
      return content.count
    }
  }

}
```

Template One

```
/// Supply the default "slicing" `subscript` for `RandomAccessCollection`
/// models that accept the default associated `SubSequence`,
/// `RandomAccessSlice<Self>`.
extension RandomAccessCollection where Self.SubSequence == RandomAccessSlice<Self>, Self.SubSequence.Index == Self.Index, Self.SubSequence.
  IndexDistance == Self.IndexDistance, Self.SubSequence.Indices == DefaultRandomAccessIndices<RandomAccessSlice<Self>>, Self.SubSequence.Iterator ==
  IndexingIterator<RandomAccessSlice<Self>>, Self.SubSequence.SubSequence == RandomAccessSlice<Self>, Self.SubSequence._Element == Self._Element,
  Self.SubSequence.Indices.Index == Self.Index, Self.SubSequence.Indices.IndexDistance == Int, Self.SubSequence.Indices.Indices ==
  DefaultRandomAccessIndices<RandomAccessSlice<Self>>, Self.SubSequence.Indices.Iterator == IndexingIterator<DefaultRandomAccessIndices<
  RandomAccessSlice<Self>>>, Self.SubSequence.Indices.SubSequence == DefaultRandomAccessIndices<RandomAccessSlice<Self>>, Self.SubSequence.Indices.
  _Element == Self.Index, Self.SubSequence.Iterator.Element == Self._Element, Self.SubSequence.SubSequence.Index == Self.Index, Self.SubSequence.
  SubSequence.IndexDistance == Self.IndexDistance, Self.SubSequence.SubSequence.Indices == DefaultRandomAccessIndices<RandomAccessSlice<Self>>, Self.
  SubSequence.SubSequence.Iterator == IndexingIterator<RandomAccessSlice<Self>>, Self.SubSequence.SubSequence.SubSequence == RandomAccessSlice<Self>,
  Self.SubSequence.SubSequence._Element == Self._Element, Self.SubSequence.Indices.IndexDistance.IntegerLiteralType == Int, Self.SubSequence.Indices.
  IndexDistance.Stride == Int, Self.SubSequence.Indices.IndexDistance._DisabledRangeIndex == Int._DisabledRangeIndex,
  Self.SubSequence.Indices.Indices.Index == Self.Index, Self.SubSequence.Indices.Indices.IndexDistance == Int, Self.SubSequence.Indices.Indices.
  Iterator == IndexingIterator<DefaultRandomAccessIndices<RandomAccessSlice<Self>>>, Self.SubSequence.Indices.Indices.SubSequence ==
  DefaultRandomAccessIndices<RandomAccessSlice<Self>>, Self.SubSequence.Indices.Indices._Element == Self.Index, Self.SubSequence.Indices.Iterator.
  Element == Self.Index, Self.SubSequence.Indices.SubSequence.Index == Self.Index, Self.SubSequence.Indices.SubSequence.Iterator == IndexingIterator<
  DefaultRandomAccessIndices<RandomAccessSlice<Self>>>, Self.SubSequence.Indices.SubSequence.SubSequence == DefaultRandomAccessIndices<
  RandomAccessSlice<Self>>, Self.SubSequence.Indices.SubSequence._Element == Self.Index, Self.SubSequence.SubSequence.Indices.Index == Self.Index,
  Self.SubSequence.SubSequence.Indices.IndexDistance == Int, Self.SubSequence.SubSequence.Indices.Iterator == IndexingIterator<
  DefaultRandomAccessIndices<RandomAccessSlice<Self>>>, Self.SubSequence.SubSequence.Indices.SubSequence == DefaultRandomAccessIndices<
  RandomAccessSlice<Self>>, Self.SubSequence.SubSequence.Indices._Element == Self.Index, Self.SubSequence.SubSequence.Iterator.Element == Self.
  _Element, Self.SubSequence.SubSequence.SubSequence.Index == Self.Index, Self.SubSequence.SubSequence.SubSequ
  RandomAccessSlice<Self>>, Self.SubSequence.SubSequence.SubSequence.SubSequence == RandomAccessSlice<Self>, Se
  _Element == Self._Element, Self.SubSequence.Indices.IndexDistance.Stride.IntegerLiteralType == Int, Self.SubS
  IntegerLiteralType == Int, Self.SubSequence.Indices.Indices.IndexDistance.Stride == Int, Self.SubSequence.Ind
  _DisabledRangeIndex == Int._DisabledRangeIndex, Self.SubSequence.Indices.Indices.Iterator.Element == Self.Ind
  SubSequence.Iterator.Element == Self.Index, Self.SubSequence.SubSequence.Indices.IndexDistance.IntegerLiteral
                                                  SubSequence.SubSequence.Indices.IndexDistance._Disabled
                                                  == Self.Index, Self.SubSequence.SubSequence.SubSequence
                                                  IntegerLiteralType == Int, Self.SubSequence.SubSequence


                                                  's elements.

                                                   same elements as the
                                                   `startIndex` property
                                                  particular value.

                                                  array of strings, finding
                                                  then using that index


                                                  , "Douglas", "Evarts"]
                                                  [index]

                                                  "


                                                      // 4


                                                  indices. The bounds of
                                                  ion.
                                                  mAccessSlice<Self> { get }
```





Protocols

```
protocol Summable {

  static func +(lhs: Self, rhs: Self) -> Self

}

extension Int: Summable { }
extension String: Summable { }
```

```
public func +(lhs: Int, rhs: Int) -> Int
public func +(lhs: String, rhs: String) -> String
```

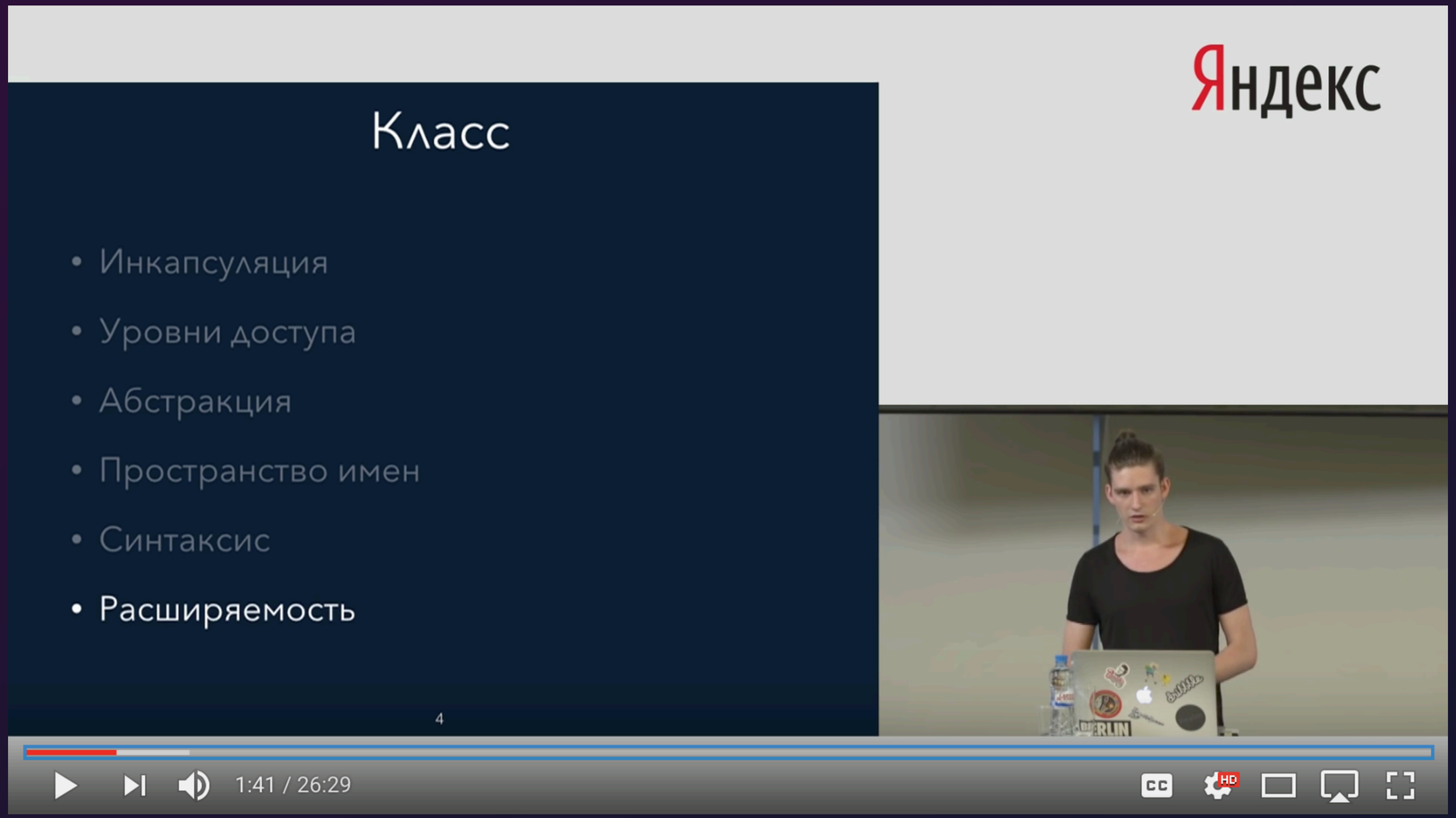
```
protocol Summator {

  associatedtype Value
  func sum(a: Value, b: Value) -> Value

}
```

```
protocol Summator {

  associatedtype Value
  func sum(a: Value, b: Value) -> Value

}

extension Summator where Value: Summable {

  func sum(a: Value, b: Value) -> Value {
    return a + b
  }

}
```

```
struct IntSummator: Summator {
  typealias Value = Int
}

let summator = IntSummator()
print(summator.sum(a: 5, b: 10)) // 15
```

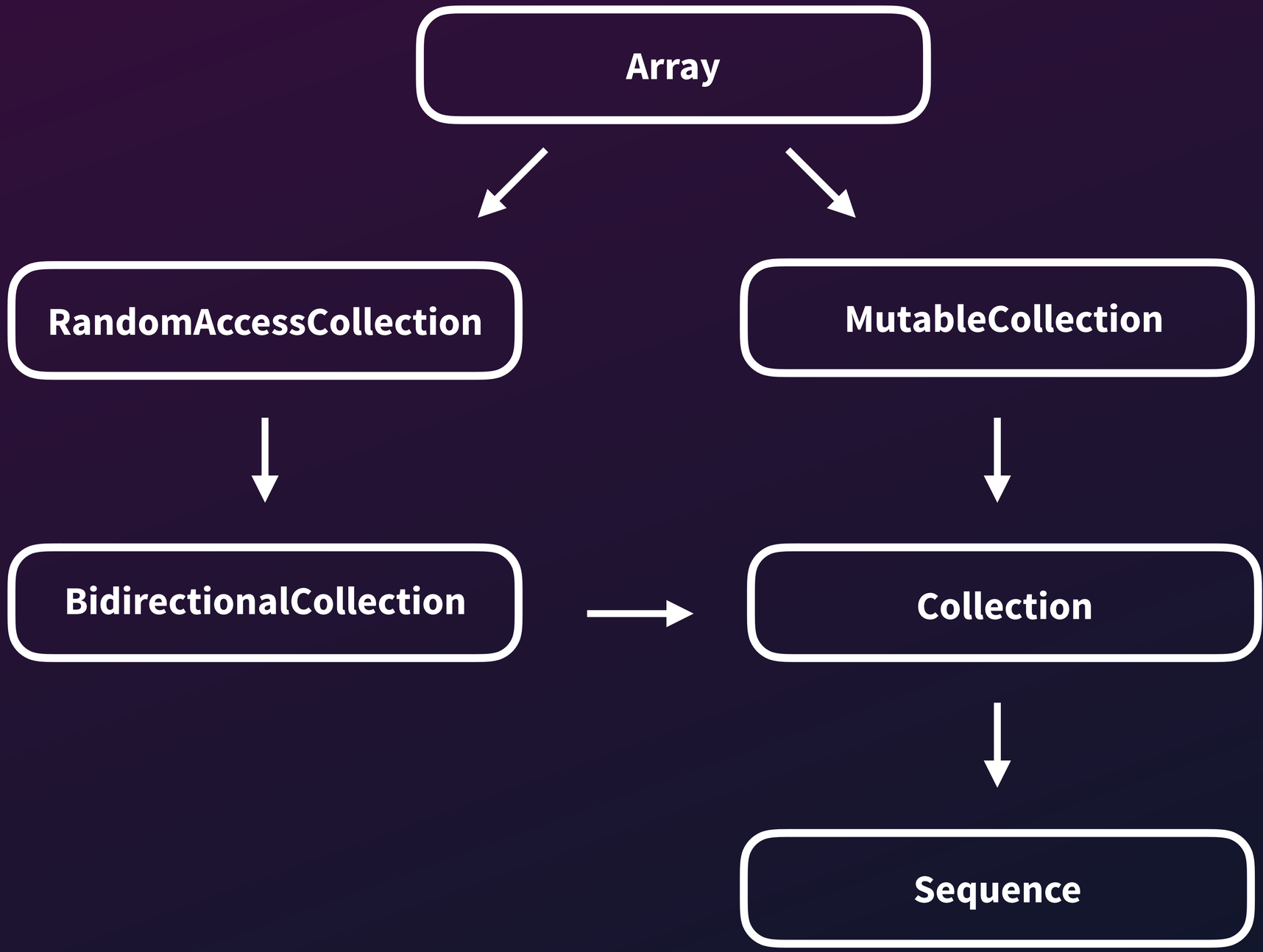
- We have Self/associatedtype for future type implementation
- We can specify rules for associatedtype
- We can extend protocols with logic
  - We can specify constraints for protocol extensions

Protocol-Oriented Programming



youtube.com/watch?v=71AS4rMrAVk

- Value Type over Reference Type
  - No inheritance
  - No mutability
- Advanced usage of protocols and it


Array
RandomAccessCollection
MutableCollection
BidirectionalCollection
Collection
Sequence

Array

RandomAccessCollection

MutableCollection

BidirectionalCollection

Collection

Sequence

ExpressibleByArrayLiteral

CustomReflectable

RangeReplaceableCollection

CustomStringConvertible

CustomDebugStringConvertible

Array

RandomAccessCollection

MutableCollection

BidirectionalCollection

Collection

Sequence

ExpressibleByArrayLiteral

CustomReflectable

RangeReplaceableCollection

CustomStringConvertible

CustomDebugStringConvertible

_ArrayProtocol

_ObjectiveCBridgeable

Encodable

Decodable

MyArrayBufferProtocol

MyPrintable

BuildableCollectionProtocol

SplittableCollection

P5

MutableCollectionAlgorithms

Array	Dictionary
RandomAccessCollection	-
MutableCollection	-
BidirectionalCollection	-
Collection	Collection
Sequence	Sequence
ExpressibleByArrayLiteral	ExpressibleByDictionaryLiteral
CustomReflectable	CustomReflectable
RangeReplaceableCollection	-
CustomStringConvertible	CustomStringConvertible
CustomDebugStringConvertible	CustomDebugStringConvertible

Array	List
RandomAccessCollection	RandomAccessCollection
MutableCollection	LazyCollectionProtocol
BidirectionalCollection	BidirectionalCollection
Collection	Collection
Sequence	Sequence
ExpressibleByArrayLiteral	ExpressibleByArrayLiteral
CustomReflectable	RealmCollection
RangeReplaceableCollection	RangeReplaceableCollection
CustomStringConvertible	CustomStringConvertible
CustomDebugStringConvertible	ThreadConfined

```swift
extension Collection where Index == Int {

  var second: Iterator.Element? {
    guard self.count > 1 else { return nil }
    return self[1]
  }

}

var array = ["A", "B", "C"]
print(array.second) // Optional("B")
```

Protocol

```
extension Collection where Index == Int {

  var second: Iterator.Element? {
    guard self.count > 1 else { return nil }
    return self[1]
  }

}

var array = ["A", "B", "C"]
print(array.second) // Optional("B")
```

Swift 3.1+

```
extension Collection where Index == Int {

  var second: Iterator.Element? {
    guard self.count > 1 else { return nil }
    return self[1]
  }

}

var array = ["A", "B", "C"]
print(array.second) // Optional("B")
```

```
extension Collection where Index == Int {

  var second: Iterator.Element? {
    guard self.count > 1 else { retu
    return self[1]
  }

}

var array = ["A", "B", "C"]
print(array.second) // Optional("B")
```

Generic Element

```
extension Collection where Index == Int {

  var second: Iterator.Element? {
    guard self.count > 1 else { return nil }
    return self[1]
  }

}

var array = ["A", "B", "C"]
print(array.second) // Optional("B")
```

Index == Int

```
subscript(position: Self.Index)
    -> Self.Iterator.Element { get }
```

```
extension Collection where Index == Int {

  var second: Iterator.Element? {
    guard self.count > 1 else { return nil }
    return self[1]
  }

}

var array = ["A", "B", "C"]
print(array.second) // Optional("B")
```

- **Follow I from SOLID**
- **Implement default behaviour in protocol extensions**
- **Implement generic based behaviour**
- **Combine multiply protocols or “inherit” them**

The **interface-segregation principle** (**ISP**) states that no client should be forced to depend on methods it does not use.

Generic

```
protocol CellViewModel {
  associatedtype CellType: UIView
  func setup(cell: CellType)
}
```

Generic

Non generic

```
protocol CellViewModel: CellViewAnyModel {
  associatedtype CellType: UIView
  func setup(cell: CellType)
}
```

associatedtype CellType: UIView

```
protocol CellViewAnyModel {
  static var cellAnyType: UIView.Type { get }
  func setupAny(cell: UIView)
}
```

func setup(cell: CellType)

```
extension CellViewModel {
  static var cellAnyType: UIView.Type {
    return CellType.self
  }

  func setupAny(cell: UIView) {
    setup(cell: cell as! CellType)
  }
}
```

- https://github.com/JohnSundell/SwiftTips
- https://developer.apple.com/swift/blog/
- https://swift.org/blog/
- https://swiftnews.curated.co
- https://www.raizlabs.com/dev/2016/12/swift-method-dispatch/
- https://github.com/ole/whats-new-in-swift-4

- https://github.com/JohnSundell/SwiftTips
- https://developer.apple.com/swift/blog/
- https://swift.org/blog/
- https://swiftnews.curated.co
- https://www.raizlabs.com/dev/2016/12/swift-method-dispatch/
- https://github.com/ole/whats-new-in-swift-4

@ZiminAlex