Strings

Что рассмотрим?

- Строки
- Протоколы
- Swift method dispatch

Strings

- Структура
- Полная поддержка Unicode
- Удобная конкатенация
- Equatable, Hashable, BidirectionalCollection, CustomReflectable, Codable

String Initialization

```swift
let someString = "Some string literal value"
```

```swift
let quotation = """
The White Rabbit put on his spectacles.  "Where shall I begin,
please your Majesty?" he asked.

"Begin at the beginning," the King said gravely, "and go on
till you come to the end; then stop."
"""
```

```swift
let singleLineString = "These are the same."
let multilineString = """
These are the same.
"""
```

```swift
let softWrappedQuotation = """
The White Rabbit put on his spectacles.  "Where shall I begin, \
please your Majesty?" he asked.

"Begin at the beginning," the King said gravely, "and go on \
till you come to the end; then stop."
"""
```

String Initialization

```swift
let wiseWords = "\"Imagination is more important than knowledge\" -
  Einstein"
// "Imagination is more important than knowledge" - Einstein
let dollarSign = "\u{24}"        // $,  Unicode scalar U+0024
let blackHeart = "\u{2665}"      // ♥,  Unicode scalar U+2665
let sparklingHeart = "\u{1F496}" // 💖, Unicode scalar U+1F496
```

```swift
let threeDoubleQuotationMarks = """
Escaping the first quotation mark \"""
Escaping all three quotation marks \"\"\"
"""
```

```swift
let threeMoreDoubleQuotationMarks = #"""
Here are three more double quotes: """
"""#
```

```swift
var emptyString = ""               // empty string literal
var anotherEmptyString = String()  // initializer syntax
// these two strings are both empty, and are equivalent to each other
```

```swift
if emptyString.isEmpty {
    print("Nothing to see here")
}
// Prints "Nothing to see here"
```

String Interpolation

```swift
let multiplier = 3
let message = "\(multiplier) times 2.5 is \(Double(multiplier) * 2.5)"
// message is "3 times 2.5 is 7.5"
```

```swift
print(#"Write an interpolated string in Swift using \(multiplier)."#)
// Prints "Write an interpolated string in Swift using \(multiplier)."
```

```swift
print(#"6 times 7 is \#(6 * 7)."#)
// Prints "6 times 7 is 42."
```

String Mutability

```swift
var variableString = "Horse"
variableString += " and carriage"
// variableString is now "Horse and carriage"

let constantString = "Highlander"
constantString += " and another Highlander"
// this reports a compile-time error - a constant string cannot be modified
```

```swift
let string1 = "hello"
let string2 = " there"
var welcome = string1 + string2
// welcome now equals "hello there"
```

```swift
var instruction = "look over"
instruction += string2
// instruction now equals "look over there"
```

Strings are value types

- Копируется при передаче в функцию или присваивании в переменную
- Thread safe
- Копирование оптимизировано компилятором

Strings are collections

```swift
for character in "Dog!🐶" {
    print(character)
}
// D
// o
// g
// !
// 🐶
```

```swift
let exclamationMark: Character = "!"
```

```swift
let catCharacters: [Character] = ["C", "a", "t", "!", "🐱"]
let catString = String(catCharacters)
print(catString)
// Prints "Cat!🐱"
```

```swift
let exclamationMark: Character = "!"
welcome.append(exclamationMark)
// welcome now equals "hello there!"
```

Strings are collections

- Набор из скалярных Unicode значений (уникальный 21-битный номер)
- Пример: U+1F425 : FRONT-FACING BABY CHICK (“🐥")
- Character - один extended grapheme cluster

```swift
let eAcute: Character = "\u{E9}"                         // é
let combinedEAcute: Character = "\u{65}\u{301}"          // e followed by ́
// eAcute is é, combinedEAcute is é
```

```swift
let precomposed: Character = "\u{D55C}"                  // 한
let decomposed: Character = "\u{1112}\u{1161}\u{11AB}"   // ᄒ, ᅡ, ᆫ
// precomposed is 한, decomposed is 한
```

Strings are collections

```swift
let unusualMenagerie = "Koala 🐨, Snail 🐌, Penguin 🐧, Dromedary 🐪"
print("unusualMenagerie has \(unusualMenagerie.count) characters")
// Prints "unusualMenagerie has 40 characters"
```

```swift
var word = "cafe"
print("the number of characters in \(word) is \(word.count)")
// Prints "the number of characters in cafe is 4"

word += "\u{301}"    // COMBINING ACUTE ACCENT, U+0301

print("the number of characters in \(word) is \(word.count)")
// Prints "the number of characters in café is 4"
```

Strings are collections

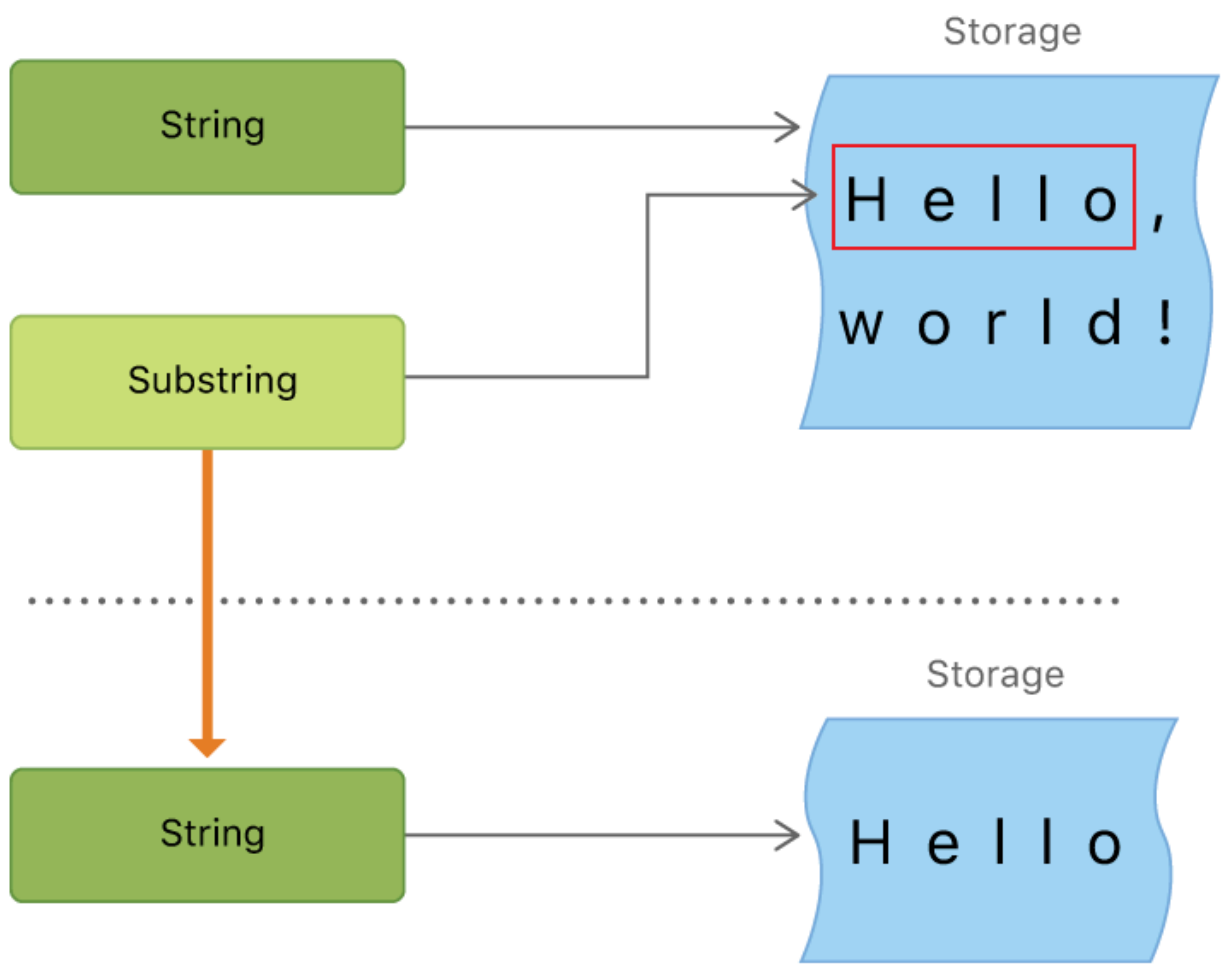
```swift
let greeting = "Guten Tag!"
greeting[greeting.startIndex]
// G
greeting[greeting.index(before: greeting.endIndex)]
// !
greeting[greeting.index(after: greeting.startIndex)]
// u
let index = greeting.index(greeting.startIndex, offsetBy: 7)
greeting[index]
// a
```

Substrings

```swift
let greeting = "Hello, world!"
let index = greeting.firstIndex(of: ",") ?? greeting.endIndex
let beginning = greeting[..<index]
// beginning is "Hello"

// Convert the result to a String for long-term storage.
let newString = String(beginning)
```

StringProtocol



Protocols

Protocols

- Является контрактом методов, пропертей и др. обязательств
- Определяет интерфейс взаимодействия
- Конформация структурами, классами, перечислениями
- Есть возможность добавить реализацию по-умолчанию
- Аналог множественного наследования

Protocol Syntax

```swift
protocol SomeProtocol {
    // protocol definition goes here
}
```

```swift
struct SomeStructure: FirstProtocol, AnotherProtocol {
    // structure definition goes here
}
```

```swift
class SomeClass: SomeSuperclass, FirstProtocol, AnotherProtocol {
    // class definition goes here
}
```

Property Requirements

```swift
protocol SomeProtocol {
    var mustBeSettable: Int { get set }
    var doesNotNeedToBeSettable: Int { get }
}
```

```swift
protocol AnotherProtocol {
    static var someTypeProperty: Int { get set }
}
```

Method Requirements

```swift
protocol SomeProtocol {
    static func someTypeMethod()
}
```

```swift
protocol RandomNumberGenerator {
    func random() -> Double
}
```

```swift
class LinearCongruentialGenerator: RandomNumberGenerator {
    var lastRandom = 42.0
    let m = 139968.0
    let a = 3877.0
    let c = 29573.0
    func random() -> Double {
        lastRandom = ((lastRandom * a + c)
            .truncatingRemainder(dividingBy:m))
        return lastRandom / m
    }
}
let generator = LinearCongruentialGenerator()
print("Here's a random number: \(generator.random())")
// Prints "Here's a random number: 0.3746499199817101"
print("And another one: \(generator.random())")
// Prints "And another one: 0.729023776863283"
```

Mutating Method Requirements

```swift
protocol Togglable {
    mutating func toggle()
}
```

```swift
enum OnOffSwitch: Togglable {
    case off, on
    mutating func toggle() {
        switch self {
        case .off:
            self = .on
        case .on:
            self = .off
        }
    }
}
var lightSwitch = OnOffSwitch.off
lightSwitch.toggle()
// lightSwitch is now equal to .on
```

Initializer Requirements

```swift
protocol SomeProtocol {
    init(someParameter: Int)
}
```

```swift
class SomeClass: SomeProtocol {
    required init(someParameter: Int) {
        // initializer implementation goes here
    }
}
```

Delegation

```swift
protocol DiceGame {
    var dice: Dice { get }
    func play()
}
protocol DiceGameDelegate: AnyObject {
    func gameDidStart(_ game: DiceGame)
    func game(_ game: DiceGame, didStartNewTurnWithDiceRoll diceRoll: Int)
    func gameDidEnd(_ game: DiceGame)
}
```

```swift
protocol SomeClassOnlyProtocol: AnyObject, SomeInheritedProtocol {
    // class-only protocol definition goes here
}
```

Adding Protocol Conformance with an Extension

```swift
protocol TextRepresentable {
    var textualDescription: String { get }
}
```

```swift
extension Dice: TextRepresentable {
    var textualDescription: String {
        return "A \(sides)-sided dice"
    }
}
```

```swift
extension Array: TextRepresentable where Element: TextRepresentable {
    var textualDescription: String {
        let itemsAsText = self.map { $0.textualDescription }
        return "[" + itemsAsText.joined(separator: ", ") + "]"
    }
}
let myDice = [d6, d12]
print(myDice.textualDescription)
// Prints "[A 6-sided dice, A 12-sided dice]"
```

Declaring Protocol Adoption with an Extension

```
struct Hamster {
    var name: String
    var textualDescription: String {
        return "A hamster named \(name)"
    }
}
extension Hamster: TextRepresentable {}
```

Protocol Inheritance

```
protocol InheritingProtocol: SomeProtocol, AnotherProtocol {
    // protocol definition goes here
}
```

```
protocol SomeClassOnlyProtocol: AnyObject, SomeInheritedProtocol {
    // class-only protocol definition goes here
}
```

Protocol Composition

```swift
protocol Named {
    var name: String { get }
}
protocol Aged {
    var age: Int { get }
}
struct Person: Named, Aged {
    var name: String
    var age: Int
}
func wishHappyBirthday(to celebrator: Named & Aged) {
    print("Happy birthday, \(celebrator.name), you're \(celebrator.age)!")
}
let birthdayPerson = Person(name: "Malcolm", age: 21)
wishHappyBirthday(to: birthdayPerson)
// Prints "Happy birthday, Malcolm, you're 21!"
```

Checking for Protocol Conformance

- is
- as?
- as!
- as

```
protocol HasArea {
    var area: Double { get }
}
```

```
class Circle: HasArea {
    let pi = 3.1415927
    var radius: Double
    var area: Double { return pi * radius * radius }
    init(radius: Double) { self.radius = radius }
}
class Country: HasArea {
    var area: Double
    init(area: Double) { self.area = area }
}
```

```
class Animal {
    var legs: Int
    init(legs: Int) { self.legs = legs }
}
```

```
let objects: [AnyObject] = [
    Circle(radius: 2.0),
    Country(area: 243_610),
    Animal(legs: 4)
]
```

```
for object in objects {
    if let objectWithArea = object as? HasArea {
        print("Area is \(objectWithArea.area)")
    } else {
        print("Something that doesn't have an area")
    }
}
// Area is 12.5663708
// Area is 243610.0
// Something that doesn't have an area
```

Optional Protocol Requirements

```swift
@objc protocol CounterDataSource {
    @objc optional func increment(forCount count: Int) -> Int
    @objc optional var fixedIncrement: Int { get }
}
```

```swift
class ThreeSource: NSObject, CounterDataSource {
    let fixedIncrement = 3
}
```

Protocol Extensions

```swift
extension RandomNumberGenerator {
    func randomBool() -> Bool {
        return random() > 0.5
    }
}
```

```swift
extension Collection where Element: Equatable {
    func allEqual() -> Bool {
        for element in self {
            if element != self.first {
                return false
            }
        }
        return true
    }
}
```

Swift Method Dispatch

Dispatch Types

- Direct/Static
- Witness table
- Virtual table
- Message

Direct/Static Dispatch

- Быстрота
- Оптимизации компилятора (inlining)
- Отсутствие полиморфизма и наследования

Witness Table

- Медленнее, чем direct dispatch
- Каждый элемент содержит таблицу под каждый протокол
- Реализует полиморфизм
- Отсутствует наследование

Witness Table

```swift
protocol Drawable {
  var size: CGSize { get }

  func draw()
  func erase()
}
```

Protocol Witness Table

Offset	0xB00	Resizable
0	0x213	originalSize
1	0x227	resize
2	0x235	resetSize

Protocol Witness Table

Offset	0xA00	Drawable
0	0x121	size.getter
1	0x124	draw
2	0x135	erase

Virtual Table

- Примерно одинаковая скорость с witness table
- Каждый наследующий элемент содержит копию таблицы методов (дополнительные затраты при компиляции)
- Новая имплементация overriden методов
- Реализует полиморфизм и наследование

Virtual Table

```swift
class Parent {
  func method1() {}
  func method2() {}
}
```

```swift
class Child: Parent {
  override func method2() {}
  func method3() {}
}
```

Offset	0xB00	Child
0	0x213	method1
1	0x227	method2🔥
2	0x235	method3

Offset	0xA00	Parent
0	0x121	method1
1	0x124	method2

Message Dispatch

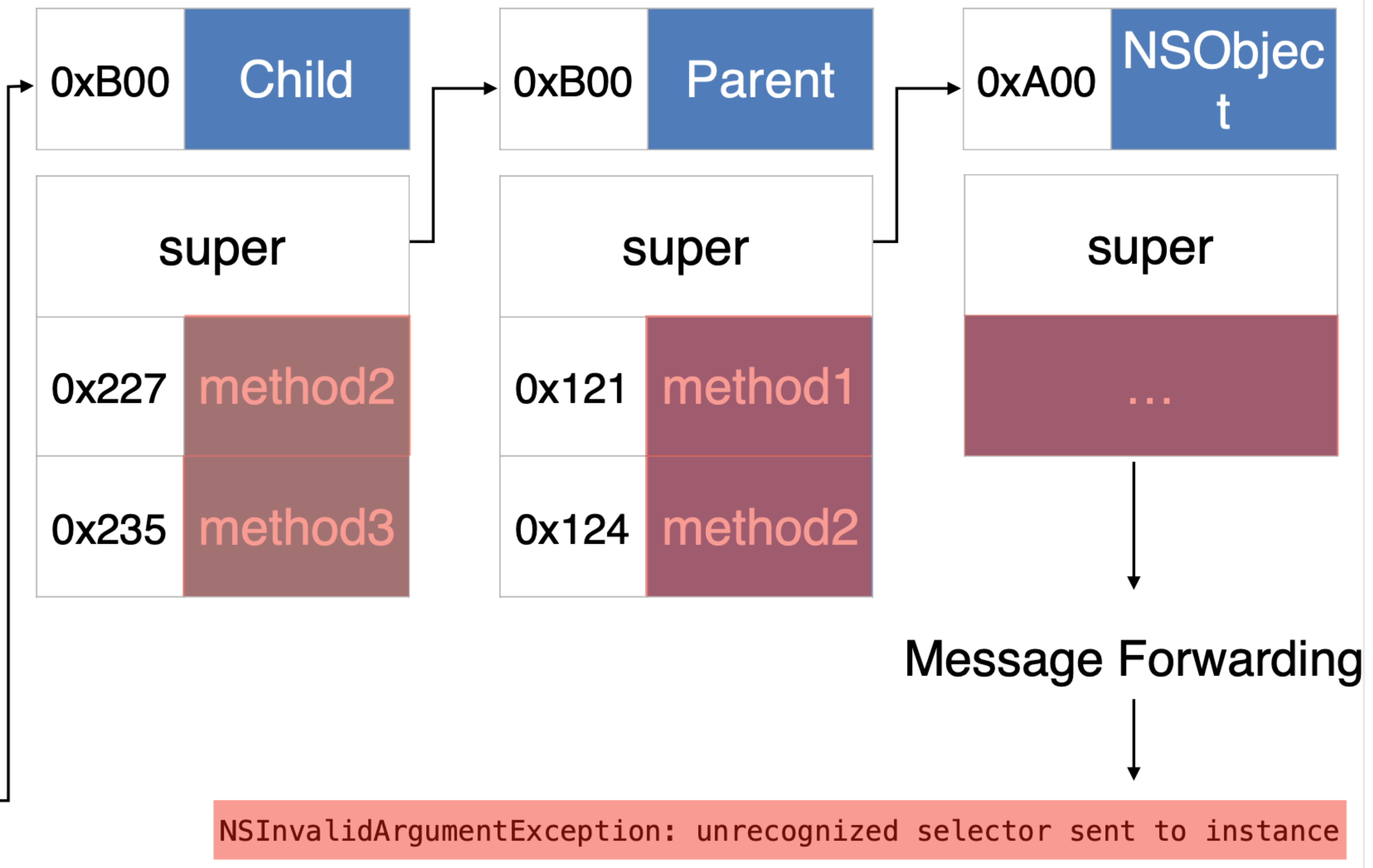
- Самый медленный тип
- Есть кеширование вызовов
- KVO
- Message Forwarding
- isa swizzling
- Реализует полиморфизм и наследование
- dynamic

Message Dispatch

```swift
@objcMembers
class Parent: NSObject {
  dynamic func method1() {}
  dynamic func method2() {}
}


@objcMembers
class Child: Parent {
  override dynamic func method2() {}
  dynamic func method3() {}
}


let child = Child()
child.method3()
child.performSelector("method5")
```



```
NSInvalidArgumentException: unrecognized selector sent to instance
```

Как управлять типами диспатчеризации

	Initial Declaration	Extension
Value Type	Static	Static
Protocol	Table	Static
Class	Table	Static
NSObject Subclass	Table	Message

Как управлять типами диспатчеризации

final	Static
dynamic	Message
@objc	Modify Objective-C Visibility

Что почитать

- Strings and Characters
- Protocols
- Доклад по диспатчеризации (взяты некоторые картинки)

Домашнее задание

- Реализовать все методы диспатчеризации
- Реализовать функцию на сложение двух чисел в представлении строк
- Числа целые положительные, оптимальное решение, нет необходимости валидировать input
- `func sum(num1: String, num2: String) -> String`
- Срок до 7 июля включительно