Лекция №2

Многопоточность. Практика

Кулаков Константин

образование

Организационная часть

- Отметиться - важно
- О чем пойдет речь в сегодняшнем занятии
  - Разбалловка и требования к проекту
  - Многопоточность (Multithreading)
  - Конкурентность (Concurrency)
  - GCD (Grand Central Dispatch) [2 Часть]
- **Оставить отзыв (после занятия)**

Требования к проекту

- Использование **GCD**
- Использование **CoreData**/Relam + **UserDefaults**
- 1+ экран на **SwiftUI** + **Combine**
- Анимация через **CoreAnimation** и/или draw
- 1+ переиспользуемый **UI-компонент**
- Покрыть **Unit** и/или **UI-тестами** 1+ экран/класс
- Реализация **диплинка** на конкретный экран (по ссылки и/или пушу)
- Наличие **кастомных** UI-элементов (bottomSheet, Popup / etc)
- Использование **GitFlow** для разработки (feature/develop/release)
- Использование **API/Firebase** для клиент-серверного взаимодействия

Дополнительно:

- CI/CD
- Верстка **кодом** (AutoLayout / Frame / Libs (pin, flex, etc))
- Публикация приложения в стор
- Собственный кэш/менеджер сети

Хронология РК

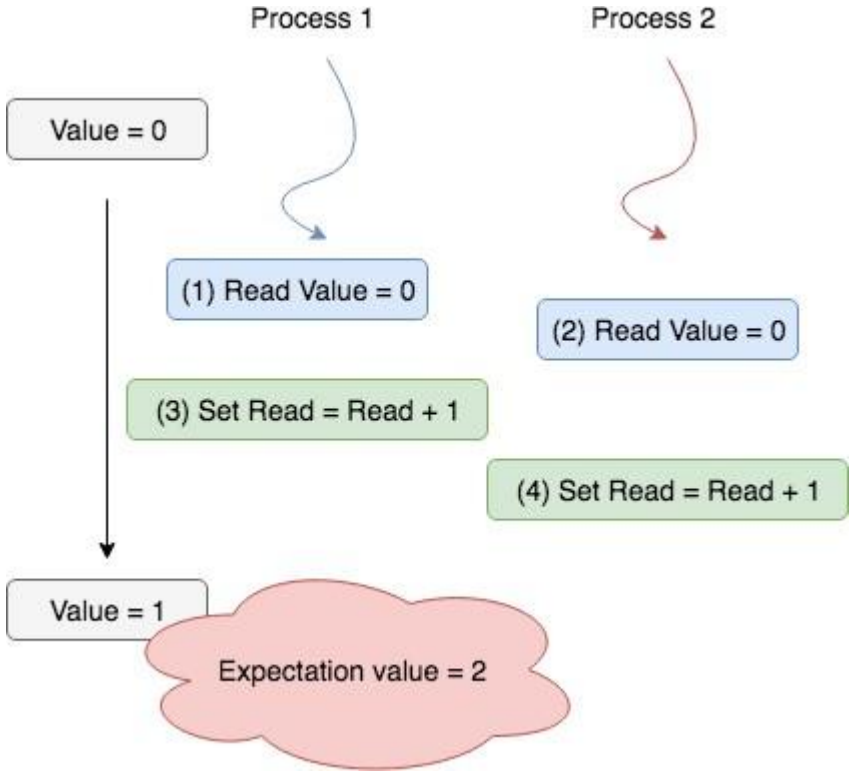
- Рубежный контроль 1 - **07 марта - 18:00**
- Рубежный контроль 2 - **11 апреля - 18:00**
- Рубежный контроль 3 - **23 мая - 18:00**

РК 1

- Разбиться на команды
- Представить презентацию проекта (идеи)
- Дизайн макет в фигме
- Стек технологий
- Контакт с ментором (ментор апрувнул макет, идею и стек)
- Открытый репозиторий и инициализированный проект
- Мемы

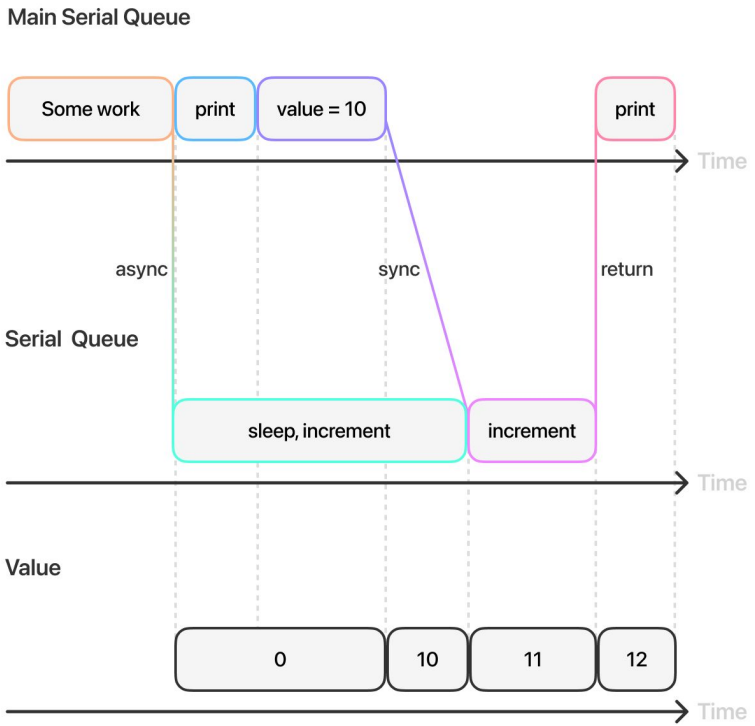
Работа с сетью. Загрузка изображений

Race condition

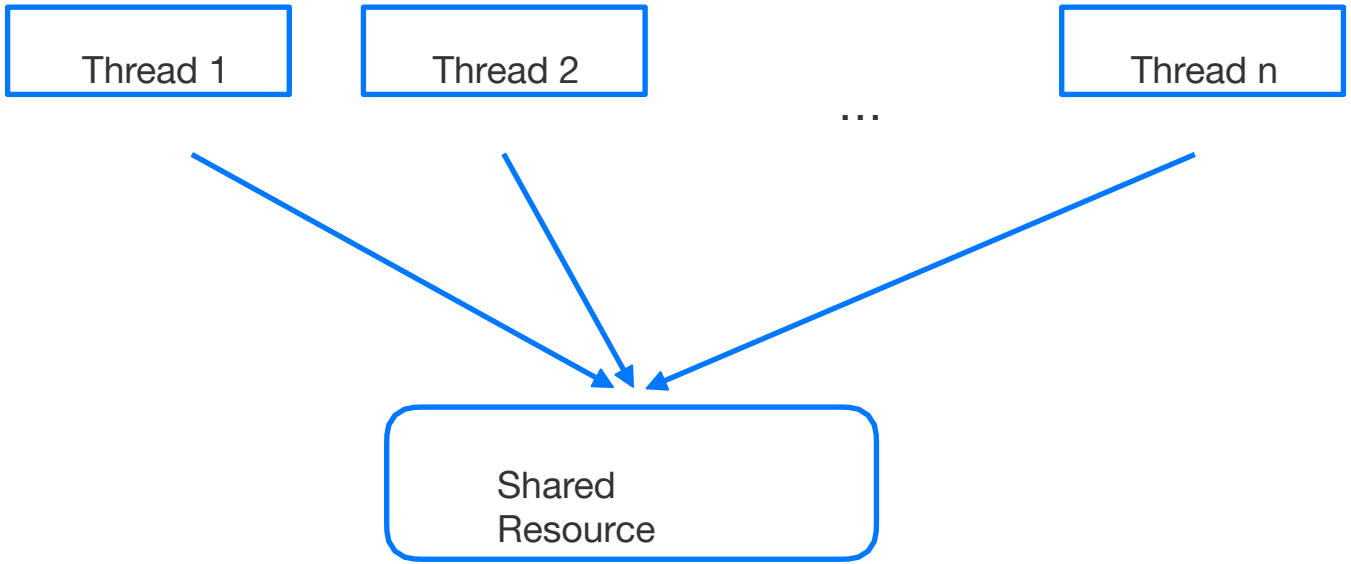


Race condition

```swift
// 1
var value: Int = 0
let serialQueue = DispatchQueue(label: "ru.serial-queue")

// 2
func increment() { value += 1 }

// 3
serialQueue.async {
    // 4
    sleep(5)
    increment()
}

// 5
print(value)

// 6
value = 10

// 7
serialQueue.sync {
    increment()
}

// 8
print(value)
```



Race condition



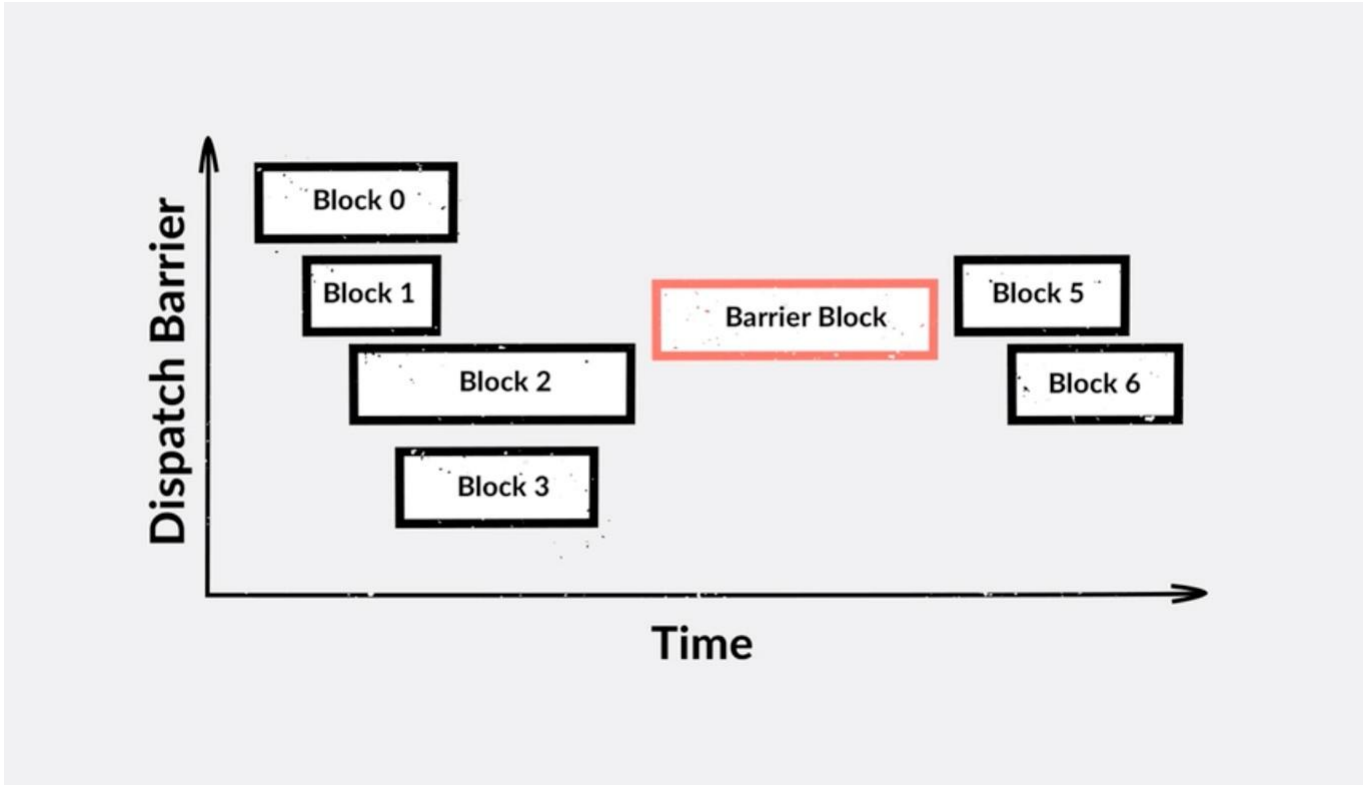
Варианты решения

- Serial queue
- Barrier

Практика. Thread Safe Array

- Serial queue
- Barrier

Dispatch barrier



Как отменить task в GCD?

DispatchWorkItem

```
let item = DispatchWorkItem {
    print("work item")
}
```

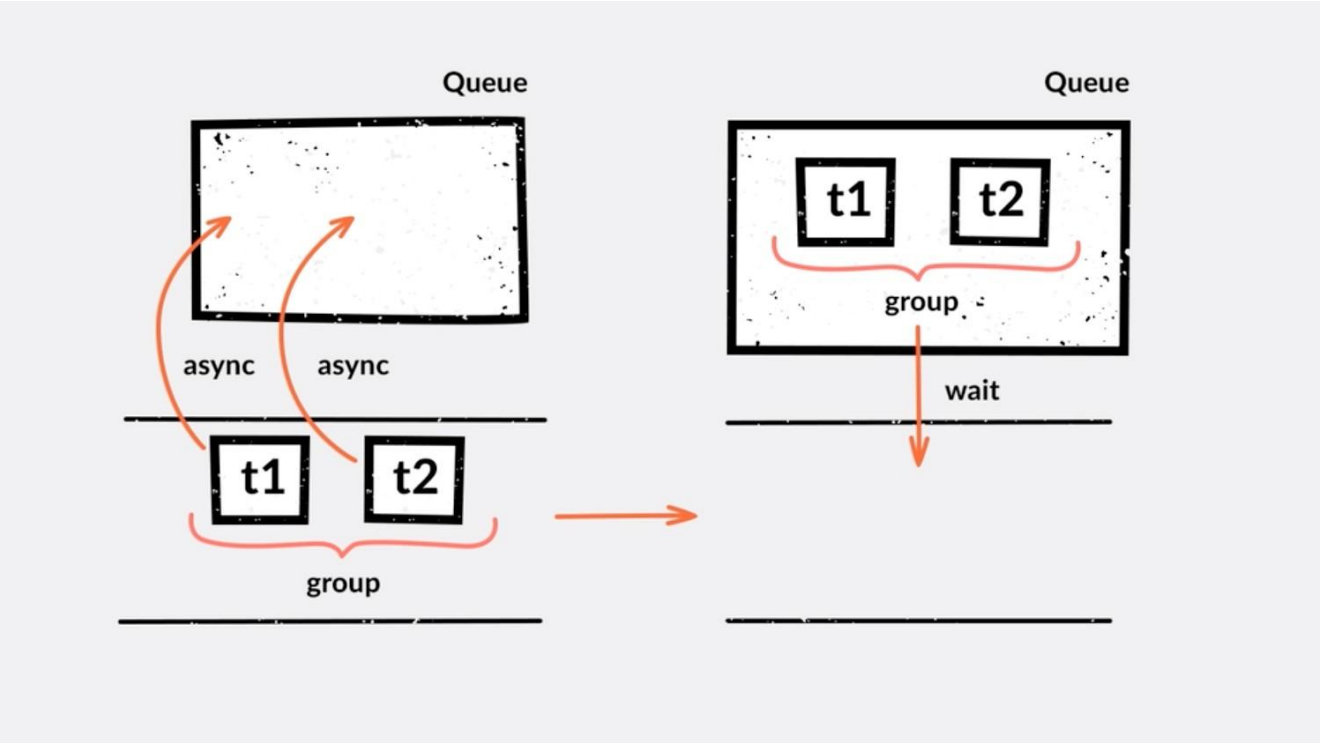
DispatchWorkItem

```
public func cancel()

public var isCancelled: Bool { get }
```

Практика. WorkItem

- Отмена отложенных операций. Лайк котенка

Dispatch group



Dispatch group

```
class DispatchGroupTest1 {
    private let group = DispatchGroup()
    private let queue = DispatchQueue(label:
"DispatchGroupTest1", attributes: .concurrent)

    func testNotify() {

        queue.async(group: group) {

            sleep(1)
            print("1")
        }
        queue.async(group: group) {

            sleep(2)
            print("2")
        }
        group.notify(queue: DispatchQueue.main) {
            print("finish all")
        }
    }
}
```

```
func loadSomeRequests() {
    let group = DispatchGroup()
    group.enter()
    api.firstRequest { [weak self] result in
        ....
        group.leave()
    }

    group.enter()
    api.request { result in
        ...

        group.leave()
    }

    group.notify(queue: .main) {
        ....
    }
}
```

Практика. DispatchGroup

- Объединение N асинхронных запросов

OperationQueue. Simple operation

```swift
// простейшая Operation
let operation1 = {
    print ("Operation 1 started")
    print ("Operation 1 finished")
}

let queue = OperationQueue()
queue.addOperation(operation1)
```

```
(()) -> ()
"Operation 1 starte
"Operation 1 finish


<NSOperationQue
<NSOperationQue
```

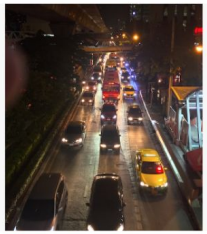
BlockOperation

Создаем `BlockOperation` для конкатенации двух строк

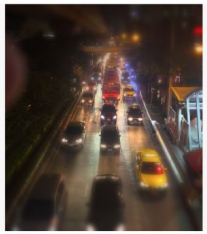
```swift
var result: String?
// DONE: Создайте и запустите concatenationOperation
let concatenationOperation = BlockOperation {
    result = "💧" + "☂️"
}
duration {
    concatenationOperation.start()
}
result
```

```
nil

<NSBlocl
()

0.00025
()

"💧☂️"
```

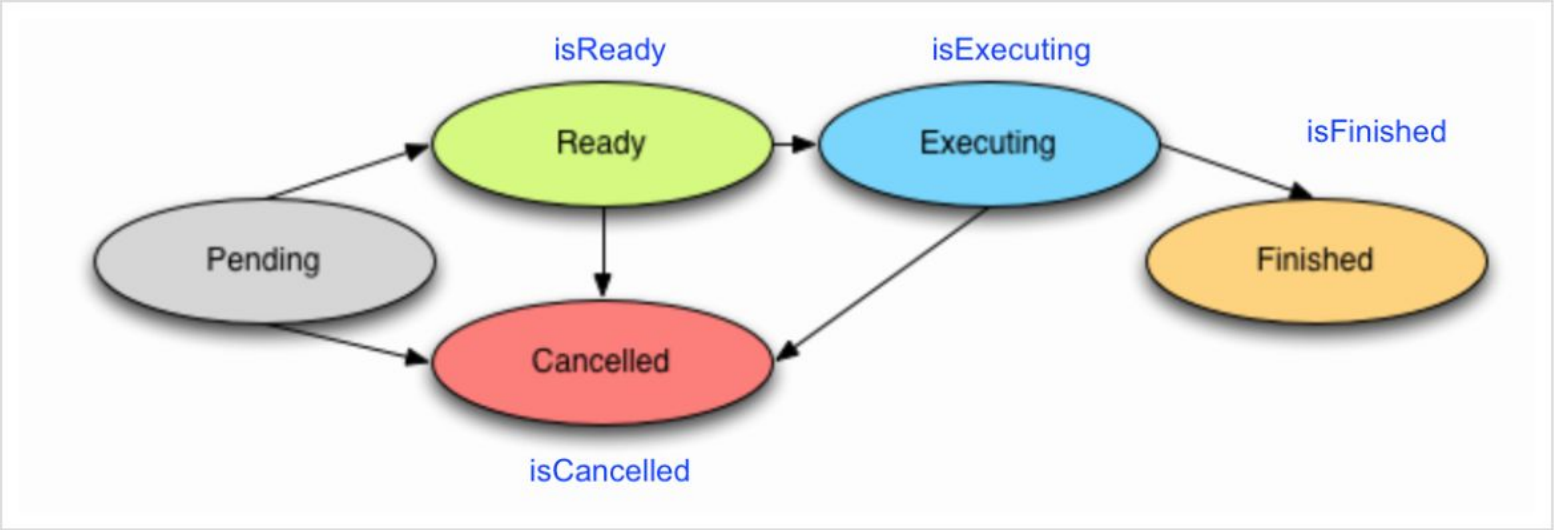
OperationQueue. Operation subclass



```
// DONE: Создайте и запустите  FilterOperation

class FilterOperation: Operation {
  var inputImage: UIImage?
  var outputImage: UIImage?

  override func main() {
    outputImage = filter(image: inputImage)
  }
}

let filterOp = FilterOperation()
filterOp.inputImage = inputImage

duration {
 filterOp.start()
}
filterOp.outputImage
```



OperationQueue. State Machine



OperationQueue. State Machine

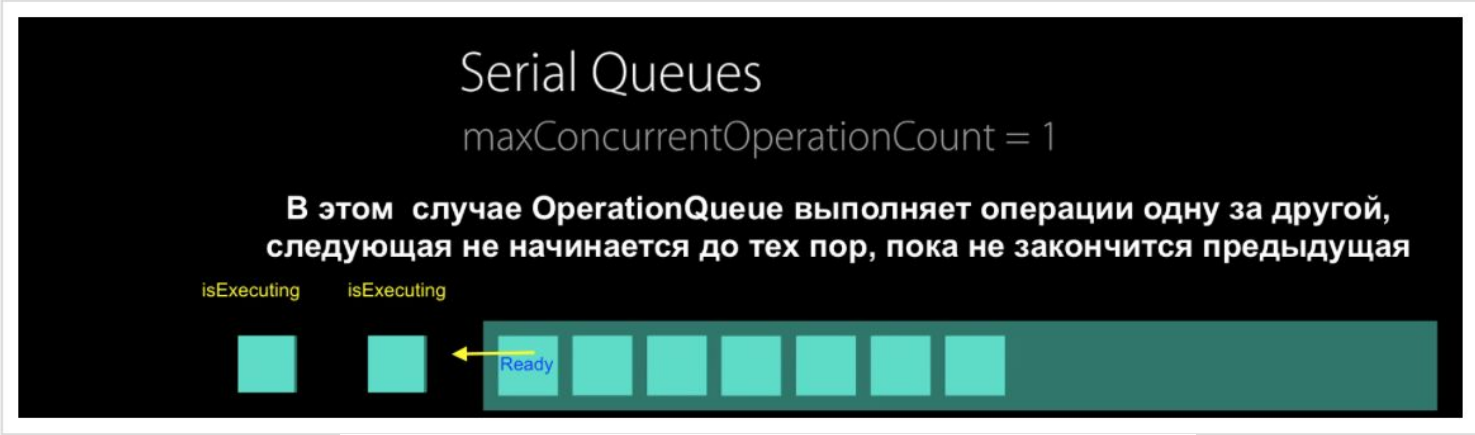
Возможные состояния операции `Operation` : `pending` (отложенная), `ready` (готова к выполнению), `executing` (выполняется), `finished` (закончена) и `cancelled` (уничтожена).

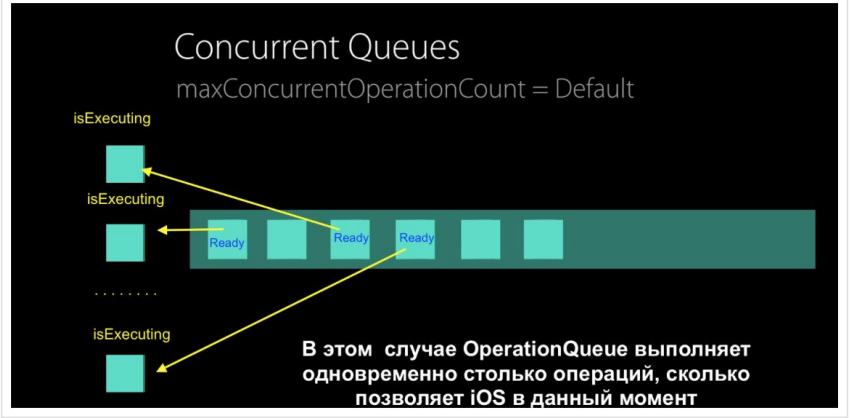
Когда вы создаете операцию `Operation` и размещаете ее на `OperationQueue` , то устанавливаете операцию в состояние `pending` (отложенная). Спустя некоторое время она принимает состояние `ready` (готова к выполнению), и в любой момент может быть отправлена на `OperationQueue` для выполнение, перейдя в состояние `executing` (выполняется), которое может длится от миллисекунд до нескольких минут или дольше. После завершения операция `Operation` переходит в финальное состояние `finished` (закончена). В любой точке этого простого «жизненного» цикла операция `Operation` может быть уничтожена и перейдет в состояние `cancelled` (уничтожена).

OperationQueue. Lifecycle

```
open class Operation : NSObject {
  open func start()
  open func main()
  open var isCancelled: Bool { get }
  open func cancel()

  open var isExecuting: Bool { get }
  open var isFinished: Bool { get }
  open var isAsynchronous: Bool { get }
  open var isReady: Bool { get }

  open var completionBlock: (() -> Swift.Void)?
  open var qualityOfService: QualityOfService
  open var name: String?
  ...
}
```

OperationQueue. Concurrency





OperationQueue. Практика

```
// DONE: Добавляем 5 операций на очередь printerQueue

duration {

  printerQueue.addOperation {  print("Каждый 🍎"); sleep(2) }
  printerQueue.addOperation {  print("Охотник 🍊");sleep(2) }
  printerQueue.addOperation {  print("Желает 🍋"); sleep(2) }
  printerQueue.addOperation {  print("Знать 🍏");  sleep(2) }
  printerQueue.addOperation {  print("Где 💎");    sleep(2) }
  printerQueue.addOperation {  print("Сидит 🚙");  sleep(2) }
  printerQueue.addOperation {  print("Фазан 🍆");  sleep(2) }

}

// DONE: Измеряем длительность всех операций
duration {
  printerQueue.waitUntilAllOperationsAreFinished()
}
```

```
0.005451977252960205
```

```
Каждый 🍎
Охотник 🍊
Желает 🍋
Знать 🍏
Где 💎
Сидит 🚙
Фазан 🍆
```

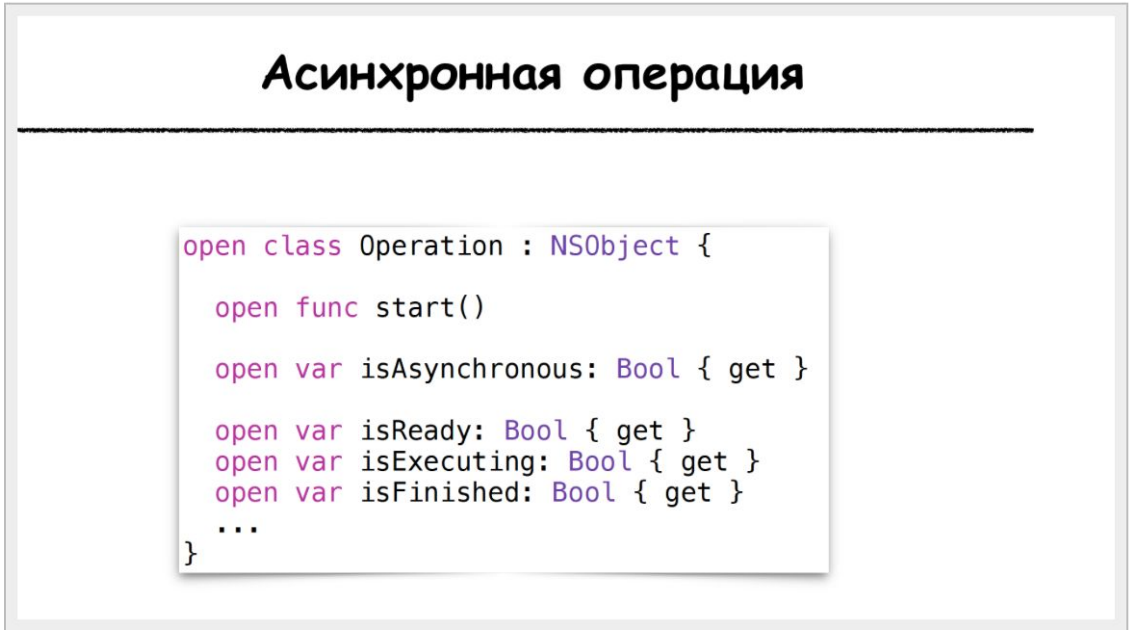
```
2.071754038333893
()
```

OperationQueue. Практика

```
// Создаем пустую очередь printerQueue
let printerQueue = OperationQueue()

// устанавливаем число одновременно выполняемых операций
printerQueue.maxConcurrentOperationCount = 2

// DONE: Добавляем 5 операций на очередь printerQueue

duration {

  printerQueue.addOperation {  print("Каждый 🍎"); sleep(2) }
  printerQueue.addOperation {  print("Охотник 🍊");sleep(2) }
  printerQueue.addOperation {  print("Желает 🍋"); sleep(2) }
  printerQueue.addOperation {  print("Знать 🍏");  sleep(2) }
  printerQueue.addOperation {  print("Где 💎");    sleep(2) }
  printerQueue.addOperation {  print("Сидит 🚙");  sleep(2) }
  printerQueue.addOperation {  print("Фазан 🍆");  sleep(2) }

}

// DONE: Измеряем длительность всех операций
duration {
  printerQueue.waitUntilAllOperationsAreFinished()
}
```

```
0.004980981349945068

(3 times)
(3 times)
(3 times)
(3 times)
(3 times)
(3 times)
(3 times)

8.183098018169403
()
```

OperationQueue. Async



Полезные ссылки

GCD и Dispatch Queues (https://www.freecodecamp.org/news/ios-concurrency/)

Ultimate GCD in Swift (https://theswiftdev.com/ultimate-grand-central-dispatch-tutorial-in-swift/)

Rambler.iOS #4: Задачи синхронизации. Классические и прикладные решения by Толстой Егор (https://www.youtube.com/watch?v=y0UQEioIgTQ)

Вопросы

Спасибо за внимание!

образование