Nomenclatura em Swift

1. CamelCase para Nomes de Variáveis e Funções:

Definição: Use CamelCase para nomes de variáveis e funções. Comece com uma letra minúscula e capitalize cada palavra subsequente.

Exemplo:

swift

var nomeDoUsuario: String

func calcularDistancia() { /\* ... \*/ }

2. PascalCase para Nomes de Tipos (Classes, Structs, Enums):

Definição: Use PascalCase para nomes de tipos (classes, structs, enums). Comece cada palavra com uma letra maiúscula.

Exemplo:

swift

class Usuario {

// ...

}

enum TipoVeiculo {

// ...

}

3. Sublinhado para Parâmetros Externos e Argumentos Internos:

Definição: Utilize sublinhados para nomes de parâmetros externos e mantenha nomes de argumentos internos descritivos.

Exemplo:

swift

func calcularArea(retanguloCom largura: Double, \_ altura: Double) -> Double {

// ...

}

4. Evite Prefixos Redundantes:

Definição: Evite usar prefixos redundantes nos nomes de suas entidades. O Swift já fornece um namespace forte.

Exemplo Ruim:

swift

struct SwiftStruct { /\* ... \*/ }

class CSharpClass { /\* ... \*/ }

5. Use Nomes Descritivos:

Definição: Escolha nomes descritivos que expressem a intenção do código. Evite abreviações ambíguas.

Exemplo:

swift

var usuariosAtivos: [Usuario]

func buscarDadosDoServidor() { /\* ... \*/ }

6. Sufixos e Prefixos para Protocolos e Delegados:

Definição: Adicione sufixos "-able" ou "-ing" a protocolos e delegados, respectivamente.

Exemplo:

swift

protocol Salvavel { /\* ... \*/ }

class ProcessadorDeDados: ProcessadorDelegado { /\* ... \*/ }

7. Utilize Verbos para Métodos:

Definição: Utilize verbos para métodos, indicando a ação que está sendo executada.

Exemplo:

swift

func calcularDistancia() { /\* ... \*/ }

func enviarMensagem() { /\* ... \*/ }  
  
Estilo de Código em Swift

1. Indentação de 4 Espaços:

Definição: Use uma indentação de 4 espaços para facilitar a legibilidade.

Exemplo:

swift

if condicao {

// código

} else {

// código

}

2. Quebras de Linha para Estruturas Lógicas:

Definição: Quebre linhas para melhorar a leitura de estruturas lógicas.

Exemplo:

swift

guard let resultado = calcularResultado(),

resultado > 0,

resultado.isPar else {

// código

}

3. Espaçamento ao Redor de Operadores:

Definição: Utilize espaçamento ao redor de operadores para melhorar a clareza.

Exemplo:

swift

let resultado = valor \* 2 + 1

4. Espaçamento Após Vírgulas:

Definição: Adicione espaçamento após vírgulas para melhorar a leitura.

Exemplo:

swift

let cores = ["vermelho", "verde", "azul"]

5. Linhas em Branco para Separar Blocos Lógicos:

Definição: Utilize linhas em branco para separar blocos lógicos de código.

Exemplo:

swift

func metodo1() {

// código

}

func metodo2() {

// código

}

6. Evite Linhas Longas:

Definição: Evite linhas de código muito longas para manter a leitura fácil.

Exemplo Ruim:

swift

let resultado = valor1 \* valor2 + valor3 - valor4 \* valor5 + valor6 \* valor7

Exemplo Melhor:

swift

let resultado = (valor1 \* valor2 + valor3 -

valor4 \* valor5 + valor6 \* valor7)

7. Maiúsculas para Palavras-Chave e Minúsculas para Nomes:

Definição: Use maiúsculas para palavras-chave da linguagem e minúsculas para nomes de variáveis, funções, etc.

Exemplo:

swift

if condicao {

metodo()

}

8. Use typealias para Tipos Complexos:

Definição: Use typealias para simplificar a leitura de tipos complexos.

Exemplo:

swift

typealias ResultadoCompleto = (valor1: Int, valor2: String)

Protocolos em Swift

1. Utilize Protocolos para Definir Contratos:

Definição: Use protocolos para definir contratos entre diferentes partes do seu código.

Exemplo:

swift

protocol Autenticavel {

func autenticar() -> Bool

}

2. Nomeie Protocolos com Substantivos Descritivos:

Definição: Dê nomes descritivos a protocolos que indiquem sua funcionalidade.

Exemplo:

swift

protocol Compartilhavel {

func compartilhar()

}

3. Utilize Extensões de Protocolo para Fornecer Implementações Padrão Opcionais:

Definição: Use extensões de protocolo para fornecer implementações padrão opcionalmente.

Exemplo:

swift

protocol Logavel {

func registrarLogin()

}

extension Logavel {

func registrarLogin() {

print("Login registrado.")

}

}

4. Utilize Protocolos para Composição:

Definição: Use protocolos para criar composições flexíveis de comportamentos.

Exemplo:

swift

protocol Carregavel {

func carregarDados()

}

protocol Exibivel {

func exibirDados()

}

struct Exibidor: Carregavel, Exibivel {

// Implementações

}

5. Evite Prefixos em Nomes de Protocolos:

Definição: Evite usar prefixos desnecessários em nomes de protocolos.

Exemplo Ruim:

swift

protocol PTReutilizavel {

// ...

}

6. Conformidade Condicional com where:

Definição: Utilize a cláusula where para impor condições de conformidade em tipos associados ou genéricos.

Exemplo:

swift

protocol Contavel {

associatedtype TipoContavel

func contar() -> Int where TipoContavel: Collection

}

7. Evite Herança Múltipla de Protocolos com o Mesmo Requisito de Associação:

Definição: Evite herdar múltiplos protocolos com o mesmo requisito de associação, pois isso pode resultar em ambiguidades.

Exemplo Ruim:

swift

protocol Logavel {

associatedtype TipoLog

func registrar(log: TipoLog)

}

protocol Rastreavel {

associatedtype TipoLog

func rastrear(log: TipoLog)

}  
  
Operadores e Closures em Swift

1. Espaçamento ao Redor de Operadores:

Definição: Adicione espaçamento ao redor de operadores para melhorar a leitura.

Exemplo:

swift

let resultado = valor \* 2 + 1

2. Utilize Operadores de União e Interseção:

Definição: Utilize os operadores + e \* para união e interseção de coleções, quando apropriado.

Exemplo:

swift

let uniao = array1 + array2

let intersecao = array1 \* array2

3. Utilize Sintaxe Clara para Closures:

Definição: Utilize a sintaxe clara de closures sempre que possível, evitando formas mais verbosas.

Exemplo:

swift

let numerosPares = numeros.filter { $0 % 2 == 0 }

4. Espaçamento em Closures:

Definição: Adicione espaçamento ao redor dos parâmetros e setas em closures para melhor legibilidade.

Exemplo:

swift

let numerosPares = numeros.filter { (numero) in

return numero % 2 == 0

}

5. Utilize in em Closures Multilinha:

Definição: Utilize in para separar a lista de parâmetros do corpo da closure em closures multilinha.

Exemplo:

swift

let numerosPares = numeros.filter { (numero) in

return numero % 2 == 0

}

6. Evite Captura Não Necessária de Self:

Definição: Evite capturar self explicitamente em closures quando não é necessário.

Exemplo Ruim:

swift

DispatchQueue.main.async {

self.atualizarUI()

}

Exemplo Melhor:

swift

DispatchQueue.main.async {

atualizarUI()

}

7. Use Nomes Descritivos para Parâmetros de Closures:

Definição: Dê nomes descritivos aos parâmetros de closures para melhorar a clareza.

Exemplo:

swift

let numerosPares = numeros.filter { (numero) in

return numero % 2 == 0

}

8. Utilize @escaping para Closures Escapáveis:

Definição: Marque closures escapáveis com @escaping quando elas são armazenadas ou saem do escopo de uma função.

Exemplo:

swift

func adicionarClosureEscapavel(completion: @escaping () -> Void) {

// ...

}  
  
Gerenciamento de Memória em Swift

1. Use ARC (Contagem Automática de Referências):

Definição: Utilize ARC para gerenciar automaticamente a alocação e desalocação de memória.

Exemplo: Não há código específico para exemplificar, já que o ARC é gerenciado pelo compilador.

2. Evite Ciclos de Referência Retida (Retain Cycles):

Definição: Evite ciclos de referência retida, pois isso pode levar a vazamentos de memória. Utilize weak ou unowned quando apropriado.

Exemplo:

swift

class ClasseA {

var instanciaB: ClasseB?

}

class ClasseB {

var instanciaA: ClasseA?

}

3. Utilize weak para Evitar Retenção Forte em Delegates:

Definição: Use weak para evitar referências fortes em delegates, prevenindo potenciais ciclos de referência.

Exemplo:

swift

protocol Delegado: AnyObject {

func eventoOcorreu()

}

class ObjetoComDelegate {

weak var delegate: Delegado?

}

4. Utilize unowned Apenas Quando Certo da Não-Retenção:

Definição: Use unowned apenas quando tiver certeza de que a referência nunca será nil durante o ciclo de vida do objeto.

Exemplo:

swift

class Pai {

var filho: Filho?

}

class Filho {

unowned let pai: Pai

init(pai: Pai) {

self.pai = pai

}

}

5. Use Listas de Captura em Closures para Evitar Retenções Forçadas:

Definição: Utilize listas de captura [weak self] ou [unowned self] em closures para evitar referências fortes e possíveis ciclos de referência.

Exemplo:

swift

someClosure = { [weak self] in

self?.fazerAlgo()

}

6. Utilize [weak self] em Closures de Delegates:

Definição: Ao implementar métodos de delegates em closures, utilize [weak self] para evitar retenção forte.

Exemplo:

swift

viewModel.delegate = { [weak self] resultado in

self?.processarResultado(resultado)

}

7. Utilize Listas de Captura [unowned self] com Cuidado:

Definição: Utilize [unowned self] apenas quando tiver certeza de que a referência nunca será nil. Caso contrário, prefira [weak self].

Exemplo:

swift

someClosure = { [unowned self] in

self.fazerAlgo()

}

Guia de Estilo para Testes em Swift

1. Nomenclatura Descritiva para Testes:

Definição: Dê nomes descritivos e significativos para seus testes. Isso ajuda na compreensão do que está sendo testado.

Exemplo:

swift

func testCalcularTotalCompra() {

// ...

}

2. Separação Lógica de Testes:

Definição: Mantenha testes relacionados agrupados logicamente em diferentes métodos ou classes de teste.

Exemplo:

swift

class TestesAutenticacao: XCTestCase {

// Testes relacionados à autenticação

}

class TestesCompra: XCTestCase {

// Testes relacionados à compra

}

3. Utilize Comentários para Descrever Testes Complexos:

Definição: Se um teste é complexo ou não é imediatamente óbvio, utilize comentários para explicar o propósito e a lógica do teste.

Exemplo:

swift

func testLogicaCompraComDesconto() {

// Este teste verifica a lógica de aplicação de desconto em uma compra

// com base em certas condições.

// ...

}

4. Teste para Condições Limite (Edge Cases):

Definição: Certifique-se de incluir testes para condições limite, como valores mínimos, máximos e situações extremas.

Exemplo:

swift

func testDivisaoPorZero() {

// ...

}

5. Mantenha Testes Rápidos e Independentes:

Definição: Testes devem ser rápidos e independentes, garantindo que possam ser executados em qualquer ordem.

Exemplo:

swift

func testMultiplicacao() {

// ...

}

func testSubtracao() {

// ...

}

6. Use Mocks e Stub para Isolar Dependências Externas:

Definição: Use mocks ou stubs para isolar o código do teste de dependências externas, garantindo a testabilidade e independência.

Exemplo:

swift

func testCarregarDadosDoServidor() {

// Use um mock para simular a resposta do servidor.

// ...

}

7. Execute Testes Regularmente:

Definição: Execute testes regularmente, preferencialmente antes de enviar ou integrar o código no repositório.

Exemplo:

bash

xcodebuild test -scheme MeuApp -destination 'platform=iOS Simulator,name=iPhone 12'

8. Escreva Testes antes de Implementar a Funcionalidade:

Definição: Pratique TDD (Desenvolvimento Orientado a Testes) escrevendo testes antes de implementar a lógica de uma nova funcionalidade.

Exemplo:

swift

func testNovaFuncionalidade() {

// Escreva o teste antes de implementar a funcionalidade.

// ...

}

9. Mantenha Testes Atualizados com Mudanças de Código:

Definição: Sempre que houver mudanças no código, certifique-se de atualizar os testes correspondentes para refletir essas alterações.

Exemplo:

swift

func testAtualizacaoUsuario() {

// Atualize o teste após realizar alterações no código de atualização de usuário.

// ...

}

Essas são algumas diretrizes para escrever testes em Swift. Adaptar essas práticas ajudará a manter um conjunto de testes robusto e eficaz para o seu projeto.