The Complete iOS Developer's Handbook

From Data Structures to System Design

Table of Contents

Part I: Foundation

Chapter 1: Swift Language Mastery

1.1 Core Language Features

Optionals: Safe Programming

swift

```swift
// Optional binding
func processUser(_ user: User?) {
    guard let user = user else { return }
    // Safe to use user here
}

// Nil coalescing
let displayName = user?.name ?? "Unknown User"

// Optional chaining
let streetName = user?.address?.street?.name
```

Generics: Type-Safe Flexibility

swift

```swift
protocol Container {
    associatedtype Item
    mutating func append(_ item: Item)
    var count: Int { get }
    subscript(i: Int) -> Item { get }
}

struct Stack<Element>: Container {
    private var items = [Element]()

    mutating func push(_ item: Element) {
        items.append(item)
    }

    mutating func pop() -> Element? {
        return items.popLast()
    }

    // Container protocol conformance
    mutating func append(_ item: Element) {
        push(item)
    }

    var count: Int {
        return items.count
    }

    subscript(i: Int) -> Element {
        return items[i]
    }
}
```

Protocols: Interface-Oriented Programming

swift

```swift
protocol Drawable {
    func draw()
}

protocol Transformable {
    mutating func transform(by matrix: CGAffineTransform)
}

// Protocol composition
typealias DrawableAndTransformable = Drawable & Transformable

struct Shape: DrawableAndTransformable {
    func draw() {
        // Drawing implementation
    }

    mutating func transform(by matrix: CGAffineTransform) {
        // Transformation implementation
    }
}
```

Extensions: Adding Functionality

```swift
extension String {
    var isEmail: Bool {
        let emailRegex = #"^[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}$"#
        return range(of: emailRegex, options: [.regularExpression, .caseInsensitive]) != nil
    }

    func truncated(to length: Int) -> String {
        return count > length ? String(prefix(length)) + "..." : self
    }
}

// Usage
let email = "user@example.com"
print(email.isEmail) // true
```

1.2 Advanced Swift Features

Result Type for Error Handling

```swift
```

```swift
enum NetworkError: Error {
    case invalidURL
    case noData
    case decodingError
}

func fetchData<T: Codable>(from url: String, type: T.Type) -> Result<T, NetworkError> {
    guard let url = URL(string: url) else {
        return .failure(.invalidURL)
    }

    // Simulated network request
    // In reality, this would be async
    return .success(/* decoded data */)
}

// Usage
let result = fetchData(from: "https://api.example.com/users", type: [User].self)
switch result {
case .success(let users):
    print("Fetched \(users.count) users")
case .failure(let error):
    print("Error: \(error)")
}
```

Property Wrappers

swift

```swift
@propertyWrapper
struct Capitalized {
    private var value: String = ""

    var wrappedValue: String {
        get { value }
        set { value = newValue.capitalized }
    }
}

struct Person {
    @Capitalized var firstName: String
    @Capitalized var lastName: String
}

var person = Person()
person.firstName = "john"
print(person.firstName) // "John"
```

Chapter 2: Memory Management & ARC

2.1 Understanding ARC

swift

```swift
class Person {
    let name: String
    init(name: String) { self.name = name }
    deinit { print("\(name) is being deinitialized") }
}

var reference1: Person?
var reference2: Person?
var reference3: Person?

reference1 = Person(name: "John Appleseed")
reference2 = reference1
reference3 = reference1

// Setting references to nil
reference1 = nil
reference2 = nil
// Person instance is still alive because reference3 still holds a strong reference
reference3 = nil
// Now "John Appleseed is being deinitialized" is printed
```

2.2 Strong Reference Cycles

swift

```swift
class Person {
    let name: String
    init(name: String) { self.name = name }
    var apartment: Apartment?
    deinit { print("\(name) is being deinitialized") }
}

class Apartment {
    let unit: String
    init(unit: String) { self.unit = unit }
    weak var tenant: Person? // weak reference to break the cycle
    deinit { print("Apartment \(unit) is being deinitialized") }
}
```

2.3 Weak and Unowned References

swift

```swift
// Weak references
class WeakExample {
    weak var delegate: SomeDelegate?

    func performAction() {
        delegate?.didPerformAction()
    }
}

// Unowned references
class Customer {
    let name: String
    var card: CreditCard?
    init(name: String) { self.name = name }
}

class CreditCard {
    let number: UInt64
    unowned let customer: Customer // Customer always exists when CreditCard exists
    init(number: UInt64, customer: Customer) {
        self.number = number
        self.customer = customer
    }
}
```

Chapter 3: Concurrency & Async Programming

3.1 Grand Central Dispatch (GCD)

swift

```swift
// Serial queue
let serialQueue = DispatchQueue(label: "com.example.serial")

// Concurrent queue
let concurrentQueue = DispatchQueue(label: "com.example.concurrent", attributes: .concurrent)

// Main queue operations
DispatchQueue.main.async {
    // Update UI
}

// Background processing
DispatchQueue.global(qos: .background).async {
    // Heavy computation
    let result = performHeavyComputation()

    DispatchQueue.main.async {
        // Update UI with result
        self.updateUI(with: result)
    }
}
```

3.2 Modern Async/Await

swift

```swift
// Async function
func fetchUserData(id: String) async throws -> User {
    let url = URL(string: "https://api.example.com/users/\(id)")!
    let (data, _) = try await URLSession.shared.data(from: url)
    return try JSONDecoder().decode(User.self, from: data)
}

// Using async/await
func loadUserProfile() async {
    do {
        let user = try await fetchUserData(id: "123")
        await MainActor.run {
            // Update UI on main thread
            self.updateUserProfile(user)
        }
    } catch {
        print("Failed to load user: \(error)")
    }
}
```

3.3 Actors for Thread Safety

swift

```swift
actor BankAccount {
    private var balance: Double = 0

    func deposit(amount: Double) {
        balance += amount
    }

    func withdraw(amount: Double) -> Bool {
        if balance >= amount {
            balance -= amount
            return true
        }
        return false
    }

    func getBalance() -> Double {
        return balance
    }
}

// Usage
let account = BankAccount()
await account.deposit(amount: 100)
let canWithdraw = await account.withdraw(amount: 50)
```

Part II: Data Structures & Algorithms

Chapter 4: Essential DSA Patterns

4.1 Two Pointers Pattern

swift

```swift
// Valid Palindrome
func isPalindrome(_ s: String) -> Bool {
    let chars = Array(s.lowercased().filter { $0.isLetter || $0.isNumber })
    var left = 0
    var right = chars.count - 1

    while left < right {
        if chars[left] != chars[right] {
            return false
        }
        left += 1
        right -= 1
    }
    return true
}

// Container With Most Water
func maxArea(_ height: [Int]) -> Int {
    var left = 0
    var right = height.count - 1
    var maxArea = 0

    while left < right {
        let area = min(height[left], height[right]) * (right - left)
        maxArea = max(maxArea, area)

        if height[left] < height[right] {
            left += 1
        } else {
            right -= 1
        }
    }
    return maxArea
}
```

4.2 Sliding Window Pattern

swift

```swift
// Longest Substring Without Repeating Characters
func lengthOfLongestSubstring(_ s: String) -> Int {
    var charIndexMap = [Character: Int]()
    var maxLength = 0
    var left = 0
    let chars = Array(s)

    for right in 0..<chars.count {
        if let index = charIndexMap[chars[right]], index >= left {
            left = index + 1
        }
        charIndexMap[chars[right]] = right
        maxLength = max(maxLength, right - left + 1)
    }
    return maxLength
}

// Minimum Window Substring
func minWindow(_ s: String, _ t: String) -> String {
    let sChars = Array(s)
    let tChars = Array(t)

    var targetCount = [Character: Int]()
    for char in tChars {
        targetCount[char, default: 0] += 1
    }

    var windowCount = [Character: Int]()
    var left = 0
    var minLen = Int.max
    var minStart = 0
    var formed = 0
    let required = targetCount.count

    for right in 0..<sChars.count {
        let char = sChars[right]
        windowCount[char, default: 0] += 1

        if let targetFreq = targetCount[char], windowCount[char] == targetFreq {
            formed += 1
        }

        while formed == required {
            if right - left + 1 < minLen {
                minLen = right - left + 1
                minStart = left
```

```
            }

            let leftChar = sChars[left]
            windowCount[leftChar]! -= 1
            if let targetFreq = targetCount[leftChar], windowCount[leftChar]! < targetFreq {
                formed -= 1
            }
            left += 1
        }
    }

    return minLen == Int.max ? "" : String(sChars[minStart..<minStart + minLen])
}
```

4.3 Fast & Slow Pointers

swift

```swift
// Linked List Cycle Detection
func hasCycle(_ head: ListNode?) -> Bool {
    var slow = head
    var fast = head

    while fast?.next != nil {
        slow = slow?.next
        fast = fast?.next?.next
        if slow === fast {
            return true
        }
    }
    return false
}

// Find Duplicate Number
func findDuplicate(_ nums: [Int]) -> Int {
    var slow = nums[0]
    var fast = nums[0]

    // Find intersection point
    repeat {
        slow = nums[slow]
        fast = nums[nums[fast]]
    } while slow != fast

    // Find entrance to cycle
    slow = nums[0]
    while slow != fast {
        slow = nums[slow]
        fast = nums[fast]
    }

    return slow
}
```

4.4 Merge Intervals

swift

```
func merge(_ intervals: [[Int]]) -> [[Int]] {
    guard intervals.count > 1 else { return intervals }

    let sortedIntervals = intervals.sorted { $0[0] < $1[0] }
    var merged = [sortedIntervals[0]]

    for i in 1..<sortedIntervals.count {
        let current = sortedIntervals[i]
        var last = merged[merged.count - 1]

        if current[0] <= last[1] {
            last[1] = max(last[1], current[1])
            merged[merged.count - 1] = last
        } else {
            merged.append(current)
        }
    }

    return merged
}

// Insert Interval
func insert(_ intervals: [[Int]], _ newInterval: [Int]) -> [[Int]] {
    var result = [[Int]]()
    var newInterval = newInterval
    var i = 0

    // Add all intervals before newInterval
    while i < intervals.count && intervals[i][1] < newInterval[0] {
        result.append(intervals[i])
        i += 1
    }

    // Merge overlapping intervals
    while i < intervals.count && intervals[i][0] <= newInterval[1] {
        newInterval[0] = min(newInterval[0], intervals[i][0])
        newInterval[1] = max(newInterval[1], intervals[i][1])
        i += 1
    }
    result.append(newInterval)

    // Add remaining intervals
    while i < intervals.count {
        result.append(intervals[i])
        i += 1
    }
```

```
    return result
}
```

4.5 Tree Traversal Patterns

swift

```swift
// Binary Tree Definition
class TreeNode {
    var val: Int
    var left: TreeNode?
    var right: TreeNode?

    init(_ val: Int) {
        self.val = val
    }
}

// DFS - Inorder Traversal
func inorderTraversal(_ root: TreeNode?) -> [Int] {
    var result = [Int]()

    func inorder(_ node: TreeNode?) {
        guard let node = node else { return }
        inorder(node.left)
        result.append(node.val)
        inorder(node.right)
    }

    inorder(root)
    return result
}

// BFS - Level Order Traversal
func levelOrder(_ root: TreeNode?) -> [[Int]] {
    guard let root = root else { return [] }

    var result = [[Int]]()
    var queue = [root]

    while !queue.isEmpty {
        let levelSize = queue.count
        var currentLevel = [Int]()

        for _ in 0..<levelSize {
            let node = queue.removeFirst()
            currentLevel.append(node.val)

            if let left = node.left {
                queue.append(left)
            }
            if let right = node.right {
                queue.append(right)
```

```
            }
        }
        result.append(currentLevel)
    }

    return result
}

// Path Sum
func hasPathSum(_ root: TreeNode?, _ targetSum: Int) -> Bool {
    guard let root = root else { return false }

    if root.left == nil && root.right == nil {
        return root.val == targetSum
    }

    let remainingSum = targetSum - root.val
    return hasPathSum(root.left, remainingSum) || hasPathSum(root.right, remainingSum)
}
```

Chapter 5: Advanced Algorithms

5.1 Dynamic Programming

swift

```swift
// Fibonacci with Memoization
func fibonacci(_ n: Int) -> Int {
    var memo = [Int: Int]()

    func fib(_ n: Int) -> Int {
        if n <= 1 { return n }
        if let cached = memo[n] { return cached }

        let result = fib(n - 1) + fib(n - 2)
        memo[n] = result
        return result
    }

    return fib(n)
}

// Coin Change
func coinChange(_ coins: [Int], _ amount: Int) -> Int {
    var dp = Array(repeating: amount + 1, count: amount + 1)
    dp[0] = 0

    for i in 1...amount {
        for coin in coins {
            if coin <= i {
                dp[i] = min(dp[i], dp[i - coin] + 1)
            }
        }
    }

    return dp[amount] > amount ? -1 : dp[amount]
}

// Longest Common Subsequence
func longestCommonSubsequence(_ text1: String, _ text2: String) -> Int {
    let chars1 = Array(text1)
    let chars2 = Array(text2)
    let m = chars1.count
    let n = chars2.count

    var dp = Array(repeating: Array(repeating: 0, count: n + 1), count: m + 1)

    for i in 1...m {
        for j in 1...n {
            if chars1[i - 1] == chars2[j - 1] {
                dp[i][j] = dp[i - 1][j - 1] + 1
            } else {
```

```
                dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])
            }
        }
    }

    return dp[m][n]
}
```

5.2 Graph Algorithms

swift

```
// Graph Representation
struct Graph {
    var adjacencyList: [Int: [Int]] = [:]

    mutating func addEdge(_ from: Int, _ to: Int) {
        adjacencyList[from, default: []].append(to)
    }

    // DFS
    func dfs(from start: Int) -> [Int] {
        var visited = Set<Int>()
        var result = [Int]()

        func dfsHelper(_ node: Int) {
            if visited.contains(node) { return }
            visited.insert(node)
            result.append(node)

            for neighbor in adjacencyList[node] ?? [] {
                dfsHelper(neighbor)
            }
        }

        dfsHelper(start)
        return result
    }

    // BFS
    func bfs(from start: Int) -> [Int] {
        var visited = Set<Int>()
        var queue = [start]
        var result = [Int]()

        while !queue.isEmpty {
            let node = queue.removeFirst()
            if visited.contains(node) { continue }

            visited.insert(node)
            result.append(node)

            for neighbor in adjacencyList[node] ?? [] {
                if !visited.contains(neighbor) {
                    queue.append(neighbor)
                }
            }
        }
```

```
        return result
    }
}

// Dijkstra's Algorithm
func dijkstra(_ graph: [Int: [(Int, Int)]], start: Int) -> [Int: Int] {
    var distances = [Int: Int]()
    var visited = Set<Int>()
    var priorityQueue = [(distance: Int, node: Int)]()

    // Initialize distances
    for node in graph.keys {
        distances[node] = Int.max
    }
    distances[start] = 0
    priorityQueue.append((0, start))

    while !priorityQueue.isEmpty {
        priorityQueue.sort { $0.distance < $1.distance }
        let (currentDistance, currentNode) = priorityQueue.removeFirst()

        if visited.contains(currentNode) { continue }
        visited.insert(currentNode)

        for (neighbor, weight) in graph[currentNode] ?? [] {
            let newDistance = currentDistance + weight
            if newDistance < distances[neighbor] ?? Int.max {
                distances[neighbor] = newDistance
                priorityQueue.append((newDistance, neighbor))
            }
        }
    }

    return distances
}
```

5.3 String Algorithms

swift

```swift
// KMP Algorithm for Pattern Matching
func kmpSearch(_ text: String, _ pattern: String) -> [Int] {
    let textArray = Array(text)
    let patternArray = Array(pattern)
    let n = textArray.count
    let m = patternArray.count

    if m == 0 { return [] }

    // Build LPS array
    var lps = Array(repeating: 0, count: m)
    var len = 0
    var i = 1

    while i < m {
        if patternArray[i] == patternArray[len] {
            len += 1
            lps[i] = len
            i += 1
        } else {
            if len != 0 {
                len = lps[len - 1]
            } else {
                lps[i] = 0
                i += 1
            }
        }
    }

    // Search for pattern
    var matches = [Int]()
    i = 0
    var j = 0

    while i < n {
        if patternArray[j] == textArray[i] {
            i += 1
            j += 1
        }

        if j == m {
            matches.append(i - j)
            j = lps[j - 1]
        } else if i < n && patternArray[j] != textArray[i] {
            if j != 0 {
                j = lps[j - 1]
```

```
            } else {
                i += 1
            }
        }
    }

    return matches
}

// Rabin-Karp Algorithm
func rabinKarp(_ text: String, _ pattern: String) -> [Int] {
    let textArray = Array(text)
    let patternArray = Array(pattern)
    let n = textArray.count
    let m = patternArray.count
    let prime = 101
    let base = 256

    var patternHash = 0
    var textHash = 0
    var h = 1
    var matches = [Int]()

    // Calculate h = pow(base, m-1) % prime
    for _ in 0..<m-1 {
        h = (h * base) % prime
    }

    // Calculate hash for pattern and first window
    for i in 0..<m {
        patternHash = (base * patternHash + Int(patternArray[i].asciiValue!)) % prime
        textHash = (base * textHash + Int(textArray[i].asciiValue!)) % prime
    }

    // Slide pattern over text
    for i in 0...(n - m) {
        if patternHash == textHash {
            // Check character by character
            var match = true
            for j in 0..<m {
                if textArray[i + j] != patternArray[j] {
                    match = false
                    break
                }
            }
            if match {
                matches.append(i)
```

```
            }
        }

        // Calculate hash for next window
        if i < n - m {
            textHash = (base * (textHash - Int(textArray[i].asciiValue!) * h) + Int(textArray[i + m].asciiValue!)) % prime
            if textHash < 0 {
                textHash += prime
            }
        }
    }

    return matches
}
```

Part III: Design Principles

Chapter 7: SOLID Principles

7.1 Single Responsibility Principle (SRP)

swift

```swift
// Bad: Multiple responsibilities
class UserManager {
    func createUser(_ userData: [String: Any]) -> User {
        // Create user
    }

    func validateEmail(_ email: String) -> Bool {
        // Email validation logic
    }

    func sendWelcomeEmail(_ user: User) {
        // Email sending logic
    }

    func saveUserToDatabase(_ user: User) {
        // Database operations
    }
}

// Good: Single responsibility
class UserFactory {
    func createUser(_ userData: [String: Any]) -> User {
        // Create user
    }
}

class EmailValidator {
    func isValid(_ email: String) -> Bool {
        // Email validation logic
    }
}

class EmailService {
    func sendWelcomeEmail(_ user: User) {
        // Email sending logic
    }
}

class UserRepository {
    func save(_ user: User) {
        // Database operations
    }
}
```

7.2 Open/Closed Principle (OCP)

swift

```swift
// Bad: Modification required for new shapes
class AreaCalculator {
    func calculateArea(_ shapes: [Any]) -> Double {
        var totalArea = 0.0
        for shape in shapes {
            if let rectangle = shape as? Rectangle {
                totalArea += rectangle.width * rectangle.height
            } else if let circle = shape as? Circle {
                totalArea += Double.pi * circle.radius * circle.radius
            }
            // Need to modify this method for new shapes
        }
        return totalArea
    }
}

// Good: Open for extension, closed for modification
protocol Shape {
    func area() -> Double
}

class Rectangle: Shape {
    let width: Double
    let height: Double

    init(width: Double, height: Double) {
        self.width = width
        self.height = height
    }

    func area() -> Double {
        return width * height
    }
}

class Circle: Shape {
    let radius: Double

    init(radius: Double) {
        self.radius = radius
    }

    func area() -> Double {
        return Double.pi * radius * radius
    }
}
```

```
class AreaCalculator {
    func calculateArea(_ shapes: [Shape]) -> Double {
        return shapes.reduce(0) { $0 + $1.area() }
    }
}
```

7.3 Liskov Substitution Principle (LSP)

swift

```swift
// Bad: Violates LSP
class Bird {
    func fly() {
        // Flying logic
    }
}

class Penguin: Bird {
    override func fly() {
        fatalError("Penguins can't fly!")
    }
}

// Good: Follows LSP
protocol Bird {
    func move()
}

protocol FlyingBird: Bird {
    func fly()
}

class Sparrow: FlyingBird {
    func move() {
        fly()
    }

    func fly() {
        // Flying logic
    }
}

class Penguin: Bird {
    func move() {
        swim()
    }

    func swim() {
        // Swimming logic
    }
}
```

7.4 Interface Segregation Principle (ISP)

swift

```swift
// Bad: Fat interface
protocol Worker {
    func work()
    func eat()
    func sleep()
}

class HumanWorker: Worker {
    func work() { /* work implementation */ }
    func eat() { /* eat implementation */ }
    func sleep() { /* sleep implementation */ }
}

class RobotWorker: Worker {
    func work() { /* work implementation */ }
    func eat() { /* Robots don't eat! */ }
    func sleep() { /* Robots don't sleep! */ }
}

// Good: Segregated interfaces
protocol Workable {
    func work()
}

protocol Eatable {
    func eat()
}

protocol Sleepable {
    func sleep()
}

class HumanWorker: Workable, Eatable, Sleepable {
    func work() { /* work implementation */ }
    func eat() { /* eat implementation */ }
    func sleep() { /* sleep implementation */ }
}

class RobotWorker: Workable {
    func work() { /* work implementation */ }
}
```

7.5 Dependency Inversion Principle (DIP)

swift

```
// Bad: High-level module depends on low-level module
class MySQLDatabase {
    func save(_ data: String) {
        // MySQL specific save logic
    }
}

class UserService {
    private let database = MySQLDatabase()

    func saveUser(_ user: User) {
        database.save(user.description)
    }
}

// Good: Both depend on abstraction
protocol Database {
    func save(_ data: String)
}

class MySQLDatabase: Database {
    func save(_ data: String) {
        // MySQL specific save logic
    }
}

class PostgreSQLDatabase: Database {
    func save(_ data: String) {
        // PostgreSQL specific save logic
    }
}

class UserService {
    private let database: Database

    init(database: Database) {
        self.database = database
    }

    func saveUser(_ user: User) {
        database.save(user.description)
    }
}
```

Chapter 8: Design Patterns

8.1 Creational Patterns

Singleton Pattern

swift

```swift
class DatabaseManager {
    static let shared = DatabaseManager()
    private init() {}

    private var connections = [String: Connection]()

    func getConnection(for database: String) -> Connection {
        if let connection = connections[database] {
            return connection
        }

        let connection = Connection(database: database)
        connections[database] = connection
        return connection
    }
}

// Thread-safe singleton
class ThreadSafeSingleton {
    static let shared: ThreadSafeSingleton = {
        let instance = ThreadSafeSingleton()
        // Setup code
        return instance
    }()

    private init() {}
}
```

Factory Pattern

swift

```swift
protocol Vehicle {
    func start()
    func stop()
}

class Car: Vehicle {
    func start() { print("Car started") }
    func stop() { print("Car stopped") }
}

class Motorcycle: Vehicle {
    func start() { print("Motorcycle started") }
    func stop() { print("Motorcycle stopped") }
}

class Truck: Vehicle {
    func start() { print("Truck started") }
    func stop() { print("Truck stopped") }
}

// Factory
class VehicleFactory {
    enum VehicleType {
        case car, motorcycle, truck
    }

    static func createVehicle(type: VehicleType) -> Vehicle {
        switch type {
        case .car:
            return Car()
        case .motorcycle:
            return Motorcycle()
        case .truck:
            return Truck()
        }
    }
}

// Usage
let car = VehicleFactory.createVehicle(type: .car)
car.start()
```

Builder Pattern

swift

```
class Computer {
    let cpu: String
    let ram: String
    let storage: String
    let gpu: String?
    let bluetooth: Bool
    let wifi: Bool

    init(cpu: String, ram: String, storage: String, gpu: String?, bluetooth: Bool, wifi: Bool) {
        self.cpu = cpu
        self.ram = ram
        self.storage = storage
        self.gpu = gpu
        self.bluetooth = bluetooth
        self.wifi = wifi
    }
}

class ComputerBuilder {
    private var cpu: String = ""
    private var ram: String = ""
    private var storage: String = ""
    private var gpu: String?
    private var bluetooth: Bool = false
    private var wifi: Bool = false

    func setCPU(_ cpu: String) -> ComputerBuilder {
        self.cpu = cpu
        return self
    }

    func setRAM(_ ram: String) -> ComputerBuilder {
        self.ram = ram
        return self
    }

    func setStorage(_ storage: String) -> ComputerBuilder {
        self.storage = storage
        return self
    }

    func setGPU(_ gpu: String) -> ComputerBuilder {
        self.gpu = gpu
        return self
    }
```

```swift
    func setBluetooth(_ bluetooth: Bool) -> ComputerBuilder {
        self.bluetooth = bluetooth
        return self
    }

    func setWifi(_ wifi: Bool) -> ComputerBuilder {
        self.wifi = wifi
        return self
    }

    func build() -> Computer {
        return Computer(cpu: cpu, ram: ram, storage: storage, gpu: gpu, bluetooth: bluetooth, wifi: wifi)
    }
}

// Usage
let computer = ComputerBuilder()
    .setCPU("Intel i7")
    .setRAM("16GB")
    .setStorage("1TB SSD")
    .setGPU("RTX 3080")
    .setBluetooth(true)
    .setWifi(true)
    .build()
```

8.2 Structural Patterns

Adapter Pattern

swift

```swift
// Legacy API
class LegacyPrinter {
    func printOldFormat(_ text: String) {
        print("Legacy: \(text)")
    }
}

// New interface
protocol ModernPrinter {
    func print(_ document: Document)
}

struct Document {
    let content: String
}

// Adapter
class PrinterAdapter: ModernPrinter {
    private let legacyPrinter: LegacyPrinter

    init(legacyPrinter: LegacyPrinter) {
        self.legacyPrinter = legacyPrinter
    }

    func print(_ document: Document) {
        legacyPrinter.printOldFormat(document.content)
    }
}

// Usage
let legacyPrinter = LegacyPrinter()
let adapter = PrinterAdapter(legacyPrinter: legacyPrinter)
adapter.print(Document(content: "Hello World"))
```

Decorator Pattern

swift

```swift
protocol Coffee {
    func cost() -> Double
    func description() -> String
}

class SimpleCoffee: Coffee {
    func cost() -> Double {
        return 2.0
    }

    func description() -> String {
        return "Simple coffee"
    }
}

class CoffeeDecorator: Coffee {
    private let coffee: Coffee

    init(coffee: Coffee) {
        self.coffee = coffee
    }

    func cost() -> Double {
        return coffee.cost()
    }

    func description() -> String {
        return coffee.description()
    }
}

class MilkDecorator: CoffeeDecorator {
    override func cost() -> Double {
        return super.cost() + 0.5
    }

    override func description() -> String {
        return super.description() + ", milk"
    }
}

class SugarDecorator: CoffeeDecorator {
    override func cost() -> Double {
        return super.cost() + 0.2
    }
```

```
    override func description() -> String {
        return super.description() + ", sugar"
    }
}

// Usage
let coffee = SimpleCoffee()
let coffeeWithMilk = MilkDecorator(coffee: coffee)
let coffeeWithMilkAndSugar = SugarDecorator(coffee: coffeeWithMilk)
print("\(coffeeWithMilkAndSugar.description()) costs $\(coffeeWithMilkAndSugar.cost())")
```

Facade Pattern

swift

```swift
// Complex subsystem
class AudioEngine {
    func initialize() { print("Audio engine initialized") }
    func playSound(_ sound: String) { print("Playing: \(sound)") }
}

class VideoEngine {
    func initialize() { print("Video engine initialized") }
    func playVideo(_ video: String) { print("Playing: \(video)") }
}

class InputHandler {
    func initialize() { print("Input handler initialized") }
    func handleInput() { print("Handling input") }
}

// Facade
class GameEngine {
    private let audioEngine = AudioEngine()
    private let videoEngine = VideoEngine()
    private let inputHandler = InputHandler()

    func startGame() {
        audioEngine.initialize()
        videoEngine.initialize()
        inputHandler.initialize()
        print("Game started!")
    }

    func playMedia(_ audio: String, _ video: String) {
        audioEngine.playSound(audio)
        videoEngine.playVideo(video)
    }
}

// Usage
let game = GameEngine()
game.startGame()
game.playMedia("background.mp3", "intro.mp4")
```

8.3 Behavioral Patterns

Observer Pattern

swift

```swift
protocol Observer: AnyObject {
    func update(_ message: String)
}

protocol Observable {
    func addObserver(_ observer: Observer)
    func removeObserver(_ observer: Observer)
    func notifyObservers(_ message: String)
}

class NewsAgency: Observable {
    private var observers = [Observer]()
    private var news: String = "" {
        didSet {
            notifyObservers(news)
        }
    }

    func addObserver(_ observer: Observer) {
        observers.append(observer)
    }

    func removeObserver(_ observer: Observer) {
        observers.removeAll { $0 === observer }
    }

    func notifyObservers(_ message: String) {
        observers.forEach { $0.update(message) }
    }

    func setNews(_ news: String) {
        self.news = news
    }
}

class NewsChannel: Observer {
    private let name: String

    init(name: String) {
        self.name = name
    }

    func update(_ message: String) {
        print("[\(name)] Breaking news: \(message)")
    }
}
```

```swift
// Usage
let agency = NewsAgency()
let cnn = NewsChannel(name: "CNN")
let bbc = NewsChannel(name: "BBC")

agency.addObserver(cnn)
agency.addObserver(bbc)
agency.setNews("Swift 6.0 Released!")
```

Strategy Pattern

swift

```
protocol PaymentStrategy {
    func pay(_ amount: Double)
}

class CreditCardPayment: PaymentStrategy {
    private let cardNumber: String

    init(cardNumber: String) {
        self.cardNumber = cardNumber
    }

    func pay(_ amount: Double) {
        print("Paid $\(amount) using credit card ending in \(String(cardNumber.suffix(4)))")
    }
}

class PayPalPayment: PaymentStrategy {
    private let email: String

    init(email: String) {
        self.email = email
    }

    func pay(_ amount: Double) {
        print("Paid $\(amount) using PayPal account \(email)")
    }
}

class ApplePayPayment: PaymentStrategy {
    func pay(_ amount: Double) {
        print("Paid $\(amount) using Apple Pay")
    }
}

class PaymentProcessor {
    private var strategy: PaymentStrategy

    init(strategy: PaymentStrategy) {
        self.strategy = strategy
    }

    func setStrategy(_ strategy: PaymentStrategy) {
        self.strategy = strategy
    }

    func processPayment(_ amount: Double) {
```

```
        strategy.pay(amount)
    }
}

// Usage
let processor = PaymentProcessor(strategy: CreditCardPayment(cardNumber: "1234567890123456"))
processor.processPayment(100.0)

processor.setStrategy(PayPalPayment(email: "user@example.com"))
processor.processPayment(75.0)
```

Command Pattern

swift

```
protocol Command {
    func execute()
    func undo()
}

class Light {
    private var isOn = false

    func turnOn() {
        isOn = true
        print("Light is ON")
    }

    func turnOff() {
        isOn = false
        print("Light is OFF")
    }
}

class LightOnCommand: Command {
    private let light: Light

    init(light: Light) {
        self.light = light
    }

    func execute() {
        light.turnOn()
    }

    func undo() {
        light.turnOff()
    }
}

class LightOffCommand: Command {
    private let light: Light

    init(light: Light) {
        self.light = light
    }

    func execute() {
        light.turnOff()
    }
```

```
    func undo() {
        light.turnOn()
    }
}

class RemoteControl {
    private var command: Command?
    private var lastCommand: Command?

    func setCommand(_ command: Command) {
        self.command = command
    }

    func pressButton() {
        command?.execute()
        lastCommand = command
    }

    func pressUndo() {
        lastCommand?.undo()
    }
}

// Usage
let light = Light()
let lightOn = LightOnCommand(light: light)
let lightOff = LightOffCommand(light: light)
let remote = RemoteControl()

remote.setCommand(lightOn)
remote.pressButton()
remote.pressUndo()
```

Chapter 9: Architectural Patterns

9.1 MVC (Model-View-Controller)

swift

```swift
// Model
struct User {
    let id: String
    let name: String
    let email: String
}

class UserModel {
    private var users: [User] = []

    func addUser(_ user: User) {
        users.append(user)
    }

    func getUsers() -> [User] {
        return users
    }

    func getUserById(_ id: String) -> User? {
        return users.first { $0.id == id }
    }
}

// View
protocol UserView: AnyObject {
    func displayUsers(_ users: [User])
    func displayError(_ message: String)
}

class UserViewController: UIViewController, UserView {
    private let controller: UserController

    init(controller: UserController) {
        self.controller = controller
        super.init(nibName: nil, bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    override func viewDidLoad() {
        super.viewDidLoad()
        controller.loadUsers()
    }
```

```
    func displayUsers(_ users: [User]) {
        // Update UI with users
    }

    func displayError(_ message: String) {
        // Show error message
    }
}

// Controller
class UserController {
    private let model: UserModel
    private weak var view: UserView?

    init(model: UserModel, view: UserView) {
        self.model = model
        self.view = view
    }

    func loadUsers() {
        let users = model.getUsers()
        view?.displayUsers(users)
    }

    func addUser(name: String, email: String) {
        let user = User(id: UUID().uuidString, name: name, email: email)
        model.addUser(user)
        loadUsers()
    }
}
```

9.2 MVVM (Model-View-ViewModel)

swift

```swift
// Model
struct Product {
    let id: String
    let name: String
    let price: Double
    let description: String
}

// Service
protocol ProductService {
    func fetchProducts() async throws -> [Product]
}

class APIProductService: ProductService {
    func fetchProducts() async throws -> [Product] {
        // API call implementation
        return []
    }
}

// ViewModel
class ProductListViewModel: ObservableObject {
    @Published var products: [Product] = []
    @Published var isLoading = false
    @Published var errorMessage: String?

    private let productService: ProductService

    init(productService: ProductService) {
        self.productService = productService
    }

    func loadProducts() {
        isLoading = true
        errorMessage = nil

        Task {
            do {
                let products = try await productService.fetchProducts()
                await MainActor.run {
                    self.products = products
                    self.isLoading = false
                }
            } catch {
                await MainActor.run {
                    self.errorMessage = error.localizedDescription
```

```
                self.isLoading = false
            }
        }
    }
}

// View (SwiftUI)
struct ProductListView: View {
    @StateObject private var viewModel: ProductListViewModel

    init(viewModel: ProductListViewModel) {
        self._viewModel = StateObject(wrappedValue: viewModel)
    }

    var body: some View {
        NavigationView {
            List(viewModel.products, id: \.id) { product in
                VStack(alignment: .leading) {
                    Text(product.name)
                        .font(.headline)
                    Text("$\(product.price, specifier: "%.2f")")
                        .font(.subheadline)
                        .foregroundColor(.secondary)
                }
            }
            .navigationTitle("Products")
            .onAppear {
                viewModel.loadProducts()
            }
            .overlay {
                if viewModel.isLoading {
                    ProgressView()
                }
            }
        }
    }
}
```

9.3 VIPER (View-Interactor-Presenter-Entity-Router)

swift

```swift
// Entity
struct UserEntity {
    let id: String
    let name: String
    let email: String
}

// Interactor
protocol UserInteractorProtocol {
    func fetchUsers() -> [UserEntity]
}

class UserInteractor: UserInteractorProtocol {
    func fetchUsers() -> [UserEntity] {
        // Data fetching logic
        return []
    }
}

// Presenter
protocol UserPresenterProtocol {
    func viewDidLoad()
    func didSelectUser(_ user: UserEntity)
}

class UserPresenter: UserPresenterProtocol {
    weak var view: UserViewProtocol?
    var interactor: UserInteractorProtocol?
    var router: UserRouterProtocol?

    func viewDidLoad() {
        let users = interactor?.fetchUsers() ?? []
        view?.showUsers(users)
    }

    func didSelectUser(_ user: UserEntity) {
        router?.navigateToUserDetail(user)
    }
}

// View
protocol UserViewProtocol: AnyObject {
    func showUsers(_ users: [UserEntity])
}

class UserViewController: UIViewController, UserViewProtocol {
```

```
    var presenter: UserPresenterProtocol?

    override func viewDidLoad() {
        super.viewDidLoad()
        presenter?.viewDidLoad()
    }

    func showUsers(_ users: [UserEntity]) {
        // Update UI
    }
}

// Router
protocol UserRouterProtocol {
    func navigateToUserDetail(_ user: UserEntity)
}

class UserRouter: UserRouterProtocol {
    weak var viewController: UIViewController?

    func navigateToUserDetail(_ user: UserEntity) {
        // Navigation logic
    }
}

// Module Builder
class UserModuleBuilder {
    static func build() -> UIViewController {
        let view = UserViewController()
        let presenter = UserPresenter()
        let interactor = UserInteractor()
        let router = UserRouter()

        view.presenter = presenter
        presenter.view = view
        presenter.interactor = interactor
        presenter.router = router
        router.viewController = view

        return view
    }
}
```

Part IV: iOS Development

Chapter 10: UIKit Fundamentals

10.1 View Controller Lifecycle

swift

```swift
class CustomViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        // View has been loaded into memory
        setupUI()
        setupConstraints()
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)
        // View is about to appear
        updateUI()
    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated)
        // View has appeared
        startAnimations()
    }

    override func viewWillDisappear(_ animated: Bool) {
        super.viewWillDisappear(animated)
        // View is about to disappear
        pauseOperations()
    }

    override func viewDidDisappear(_ animated: Bool) {
        super.viewDidDisappear(animated)
        // View has disappeared
        stopOperations()
    }

    private func setupUI() {
        view.backgroundColor = .systemBackground
        // Setup UI elements
    }

    private func setupConstraints() {
        // Setup Auto Layout constraints
    }
}
```

10.2 Auto Layout and Constraints

swift

```swift
class LayoutExampleViewController: UIViewController {

    private let titleLabel = UILabel()
    private let descriptionLabel = UILabel()
    private let actionButton = UIButton(type: .system)

    override func viewDidLoad() {
        super.viewDidLoad()
        setupViews()
        setupConstraints()
    }

    private func setupViews() {
        titleLabel.text = "Welcome"
        titleLabel.font = .systemFont(ofSize: 24, weight: .bold)
        titleLabel.textAlignment = .center

        descriptionLabel.text = "This is a description"
        descriptionLabel.font = .systemFont(ofSize: 16)
        descriptionLabel.numberOfLines = 0

        actionButton.setTitle("Action", for: .normal)
        actionButton.backgroundColor = .systemBlue
        actionButton.setTitleColor(.white, for: .normal)
        actionButton.layer.cornerRadius = 8

        [titleLabel, descriptionLabel, actionButton].forEach {
            $0.translatesAutoresizingMaskIntoConstraints = false
            view.addSubview($0)
        }
    }

    private func setupConstraints() {
        NSLayoutConstraint.activate([
            // Title Label
            titleLabel.topAnchor.constraint(equalTo: view.safeAreaLayoutGuide.topAnchor, constant: 20),
            titleLabel.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 16),
            titleLabel.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -16),

            // Description Label
            descriptionLabel.topAnchor.constraint(equalTo: titleLabel.bottomAnchor, constant: 16),
            descriptionLabel.leadingAnchor.constraint(equalTo: view.leadingAnchor, constant: 16),
            descriptionLabel.trailingAnchor.constraint(equalTo: view.trailingAnchor, constant: -16),

            // Action Button
            actionButton.topAnchor.constraint(equalTo: descriptionLabel.bottomAnchor, constant: 32),
```

```
            actionButton.centerXAnchor.constraint(equalTo: view.centerXAnchor),
            actionButton.widthAnchor.constraint(equalToConstant: 200),
            actionButton.heightAnchor.constraint(equalToConstant: 50)
        ])
    }
}
```

10.3 Table Views and Collection Views

swift

```swift
// MARK: - Table View Example
class UserTableViewController: UITableViewController {

    private var users: [User] = []

    override func viewDidLoad() {
        super.viewDidLoad()
        setupTableView()
        loadUsers()
    }

    private func setupTableView() {
        tableView.register(UITableViewCell.self, forCellReuseIdentifier: "UserCell")
        tableView.rowHeight = 60
    }

    private func loadUsers() {
        // Load users from API or database
    }

    // MARK: - Table View Data Source
    override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return users.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "UserCell", for: indexPath)
        let user = users[indexPath.row]
        cell.textLabel?.text = user.name
        cell.detailTextLabel?.text = user.email
        return cell
    }

    // MARK: - Table View Delegate
    override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.deselectRow(at: indexPath, animated: true)
        let user = users[indexPath.row]
        // Handle user selection
    }
}

// MARK: - Collection View Example
class PhotoCollectionViewController: UICollectionViewController {

    private var photos: [Photo] = []
```

```
    override func viewDidLoad() {
        super.viewDidLoad()
        setupCollectionView()
        loadPhotos()
    }

    private func setupCollectionView() {
        let layout = UICollectionViewFlowLayout()
        layout.itemSize = CGSize(width: 100, height: 100)
        layout.minimumInteritemSpacing = 8
        layout.minimumLineSpacing = 8
        collectionView.collectionViewLayout = layout

        collectionView.register(PhotoCell.self, forCellWithReuseIdentifier: "PhotoCell")
    }

    private func loadPhotos() {
        // Load photos
    }

    // MARK: - Collection View Data Source
    override func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int {
        return photos.count
    }

    override func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollec
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier: "PhotoCell", for: indexPath) as! PhotoCell
        let photo = photos[indexPath.item]
        cell.configure(with: photo)
        return cell
    }
}

class PhotoCell: UICollectionViewCell {
    private let imageView = UIImageView()

    override init(frame: CGRect) {
        super.init(frame: frame)
        setupViews()
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    private func setupViews() {
        imageView.contentMode = .scaleAspectFill
```

```
        imageView.clipsToBounds = true
        imageView.translatesAutoresizingMaskIntoConstraints = false
        contentView.addSubview(imageView)

        NSLayoutConstraint.activate([
            imageView.topAnchor.constraint(equalTo: contentView.topAnchor),
            imageView.leadingAnchor.constraint(equalTo: contentView.leadingAnchor),
            imageView.trailingAnchor.constraint(equalTo: contentView.trailingAnchor),
            imageView.bottomAnchor.constraint(equalTo: contentView.bottomAnchor)
        ])
    }

    func configure(with photo: Photo) {
        // Configure cell with photo data
    }
}
```

10.4 Navigation and Segues

swift

```swift
class NavigationExampleViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()
        setupNavigationBar()
        setupUI()
    }

    private func setupNavigationBar() {
        title = "Navigation Example"

        let addButton = UIBarButtonItem(barButtonSystemItem: .add, target: self, action: #selector(addButtonTappec
        navigationItem.rightBarButtonItem = addButton

        let backButton = UIBarButtonItem(title: "Back", style: .plain, target: self, action: #selector(backButtonTapped)
        navigationItem.leftBarButtonItem = backButton
    }

    @objc private func addButtonTapped() {
        let detailVC = DetailViewController()
        navigationController?.pushViewController(detailVC, animated: true)
    }

    @objc private func backButtonTapped() {
        navigationController?.popViewController(animated: true)
    }

    private func presentModalViewController() {
        let modalVC = ModalViewController()
        let navController = UINavigationController(rootViewController: modalVC)
        present(navController, animated: true)
    }
}
```

Chapter 11: SwiftUI Modern Development

11.1 SwiftUI Basics

swift

```swift
import SwiftUI

struct ContentView: View {
    @State private var name = ""
    @State private var isShowingDetail = false

    var body: some View {
        NavigationView {
            VStack(spacing: 20) {
                TextField("Enter your name", text: $name)
                    .textFieldStyle(RoundedBorderTextFieldStyle())
                    .padding()

                Button("Show Detail") {
                    isShowingDetail = true
                }
                .buttonStyle(PrimaryButtonStyle())

                NavigationLink("Navigate to Detail", destination: DetailView(name: name))
                    .buttonStyle(SecondaryButtonStyle())
            }
            .navigationTitle("SwiftUI Example")
            .sheet(isPresented: $isShowingDetail) {
                DetailView(name: name)
            }
        }
    }
}

struct DetailView: View {
    let name: String
    @Environment(\.dismiss) private var dismiss

    var body: some View {
        VStack {
            Text("Hello, \(name)!")
                .font(.largeTitle)
                .padding()

            Button("Dismiss") {
                dismiss()
            }
            .buttonStyle(PrimaryButtonStyle())
        }
        .navigationTitle("Detail")
        .navigationBarTitleDisplayMode(.inline)
```

```swift
    }
}

// Custom Button Styles
struct PrimaryButtonStyle: ButtonStyle {
    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.white)
            .padding()
            .background(Color.blue)
            .cornerRadius(8)
            .scaleEffect(configuration.isPressed ? 0.95 : 1.0)
    }
}

struct SecondaryButtonStyle: ButtonStyle {
    func makeBody(configuration: Configuration) -> some View {
        configuration.label
            .foregroundColor(.blue)
            .padding()
            .background(Color.clear)
            .overlay(
                RoundedRectangle(cornerRadius: 8)
                    .stroke(Color.blue, lineWidth: 2)
            )
    }
}
```

11.2 State Management

swift

```swift
// MARK: - ObservableObject
class UserStore: ObservableObject {
    @Published var users: [User] = []
    @Published var isLoading = false
    @Published var errorMessage: String?

    func fetchUsers() {
        isLoading = true
        errorMessage = nil

        // Simulate API call
        DispatchQueue.main.asyncAfter(deadline: .now() + 2) {
            self.users = [
                User(id: "1", name: "John Doe", email: "john@example.com"),
                User(id: "2", name: "Jane Smith", email: "jane@example.com")
            ]
            self.isLoading = false
        }
    }
}

// MARK: - StateObject and ObservedObject
struct UserListView: View {
    @StateObject private var userStore = UserStore()

    var body: some View {
        NavigationView {
            List(userStore.users, id: \.id) { user in
                NavigationLink(destination: UserDetailView(user: user)) {
                    VStack(alignment: .leading) {
                        Text(user.name)
                            .font(.headline)
                        Text(user.email)
                            .font(.subheadline)
                            .foregroundColor(.secondary)
                    }
                }
            }
            .navigationTitle("Users")
            .onAppear {
                userStore.fetchUsers()
            }
            .overlay {
                if userStore.isLoading {
                    ProgressView()
                        .scaleEffect(1.5)
```

```
                .frame(maxWidth: .infinity, maxHeight: .infinity)
                .background(Color.black.opacity(0.3))
            }
        }
    }
}

struct UserDetailView: View {
    let user: User
    @ObservedObject var userStore: UserStore

    var body: some View {
        VStack(alignment: .leading, spacing: 16) {
            Text(user.name)
                .font(.largeTitle)
                .fontWeight(.bold)

            Text(user.email)
                .font(.title2)
                .foregroundColor(.secondary)

            Spacer()
        }
        .padding()
        .navigationTitle("User Detail")
        .navigationBarTitleDisplayMode(.inline)
    }
}

// MARK: - Environment Objects
struct AppView: View {
    @StateObject private var userStore = UserStore()

    var body: some View {
        TabView {
            UserListView()
                .tabItem {
                    Image(systemName: "person.3")
                    Text("Users")
                }

            SettingsView()
                .tabItem {
                    Image(systemName: "gear")
                    Text("Settings")
                }
```

```
        }
        .environmentObject(userStore)
    }
}

struct SettingsView: View {
    @EnvironmentObject var userStore: UserStore

    var body: some View {
        VStack {
            Text("Settings")
                .font(.largeTitle)

            Text("Total Users: \(userStore.users.count)")
                .font(.headline)

            Button("Refresh Users") {
                userStore.fetchUsers()
            }
        }
        .padding()
    }
}
```

11.3 Custom Views and Modifiers

swift

```
// MARK: - Custom View
struct CircularProgressView: View {
    let progress: Double
    let lineWidth: CGFloat
    let backgroundColor: Color
    let foregroundColor: Color

    init(progress: Double, lineWidth: CGFloat = 10, backgroundColor: Color = .gray, foregroundColor: Color = .blue) {
        self.progress = progress
```