Объектно-ориентированное программирование в Swift

Основы ООП в Swift

Swift поддерживает ключевые концепции ООП:

- **Инкапсуляция**: скрытие данных.
- **Наследование**: расширение классов.
- **Полиморфизм**: использование общих интерфейсов.

Два основных типа сущностей:

- Классы (**class**): ссылочный тип.
- Структуры (struct): значимый тип.

Пример:

```
class Vehicle {
    var speed = 0
}
struct Point {
    var x = 0.0
}
```

Классы

Классы — это ссылочные типы, поддерживают наследование.

Пример:

```
class Car {
    var speed: Int
    init(speed: Int) {
        self.speed = speed
    }
    func accelerate() {
        speed += 10
    }
}
let myCar = Car(speed: 50)
myCar.accelerate()
print(myCar.speed) // Вывод: 60
```

Ссылочная семантика:

```
let sameCar = myCar
sameCar.speed = 70
print(myCar.speed) // Вывод: 70
```

Структуры

Структуры — это значимые типы, не поддерживают наследование.

Пример:

```
struct Point {
    var x: Double
    var y: Double
    mutating func moveBy(x: Double) {
        self.x += x
    }
}
var point = Point(x: 1.0, y: 2.0)
point.moveBy(x: 3.0)
print(point.x) // Вывод: 4.0
```

Копирование:

```
var copy = point
copy.x = 5.0
print(point.x) // Вывод: 4.0
```

Свойства и методы

Свойства с геттерами и сеттерами:

```
class Person {
    private var _name = ""
    var name: String {
        get { _name }
        set { _name = newValue.uppercased() }
    }
}
let person = Person()
person.name = "alice"
print(person.name) // Вывод: ALICE
```

Методы:

```
func greet() {
    print("Hello, \(name)")
}
person.greet() // Вывод: Hello, ALICE
```

Наследование

- Поддерживается только для классов.

Пример:

```
class Animal {
    var legs: Int
    init(legs: Int) { self.legs = legs }
    func move() { print("Moving") }
}
class Dog: Animal {
    override func move() { print("Running") }
}
let dog = Dog(legs: 4)
dog.move() // Вывод: Running
```

Модификатор **override** обязателен для переопределения.

Протоколы

- Аналог интерфейсов, определяют контракт.

Пример:

```
protocol Drivable {
    var speed: Int { get set }
    func drive()
}
class Truck: Drivable {
    var speed = 0
    func drive() { print("Truck drives at \(speed)") }
}
let truck = Truck()
truck.speed = 80
truck.drive() // Вывод: Truck drives at 80
```

Поддерживается множественное принятие протоколов.

Практическое применение

Пример моделирования **UI**-элемента:

```
protocol Viewable {
    func display()
}
class Button: Viewable {
    var title: String
    init(title: String) { self.title = title }
    func display() { print("Button: \(title)") }
}
let button = Button(title: "Click Me")
button.display() // Вывод: Button: Click Me
```

Использование в **iOS**:

- **Классы**: для контроллеров (**UIViewController**).
- **Структуры**: для данных (модели).
- **Протоколы**: для делегатов и абстракций.