SwiftUI Pills

Protocols-Oriented Programming (POP)

Simple concepts
The basics you need to know initially, to later handle the complexity of more advanced concepts.

Beginner level
Minimally complex explanations to focus on what matters.

Código de prueba
Código corto y simple para que pongas a prueba ideas y conceptos.

Test code
Short and simple code to help you try out ideas and concepts.

Clear explanations
Explanations that clarify, not complicate.

Interview questions
Questions and answers on the topic that commonly appear in technical interviews.

Knowledge in programming is like building a structure: the strength of the entire framework depends on the quality of each brick. Each unit of knowledge must be clear, solid, and understandable on its own, independent of the others. Over time, these blocks consolidate and connect creatively with others, forming a robust and harmonious structure.

Protocols are one of the fundamental concepts of Swift and hold special importance in SwiftUI. They're not just a feature for solving specific use cases but a complete programming paradigm that transforms how we understand application architecture and key project aspects, such as extensibility, reusability, and modularity.

So, what is a protocol?

A protocol in Swift is a type that defines a set of methods, properties, and other requirements that must be implemented by any class, structure, or enumeration (conformable type) that adopts it. When a type conforms to a protocol, it enters a "contract" that guarantees it will implement everything specified by the protocol.

What is the importance of protocols?

Protocols in Swift are essential for writing modular, reusable, flexible, and easy-to-maintain code. By defining behavior contracts, protocols allow us to focus on interfaces and expected behavior, regardless of the specific implementations of the types that adopt them.

The importance of protocols in Swift lies in two key aspects:

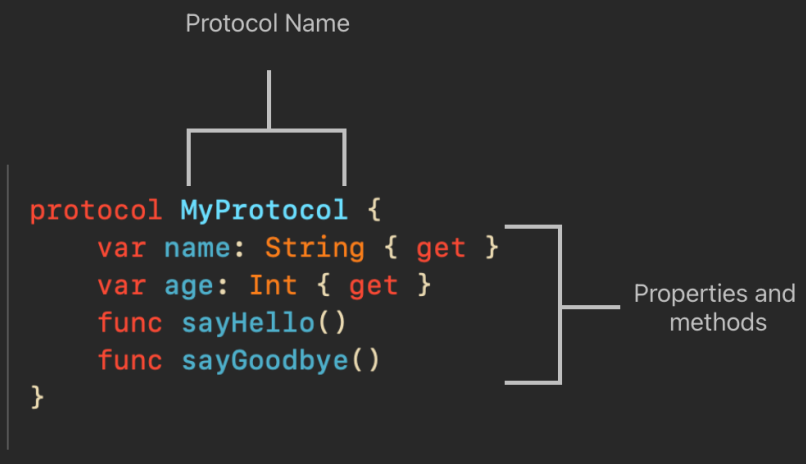
1. **Foundation of Protocol-Oriented Programming (POP):** Apple has promoted a development paradigm known as Protocol-Oriented Programming (POP), in which protocols play a central role in code design and organization. This approach encourages the composition of behaviors through protocols rather than relying exclusively on class inheritance, allowing different types to share functionality in a flexible and efficient manner.

2. **Extensive Use in the Apple Ecosystem**: Protocols are one of the most widely used resources in Apple libraries, such as Foundation and SwiftUI. The architecture of these libraries is designed to rely on protocols that facilitate interoperability and flexibility among their components. Apple actively recommends the use of protocols in application development, as they promote code clarity and scalability in complex projects.

Thanks to protocols, developers can define structures and classes that adhere to well-defined interfaces and reuse modules more effectively, promoting a more robust and adaptable architecture.

How is a protocol defined?

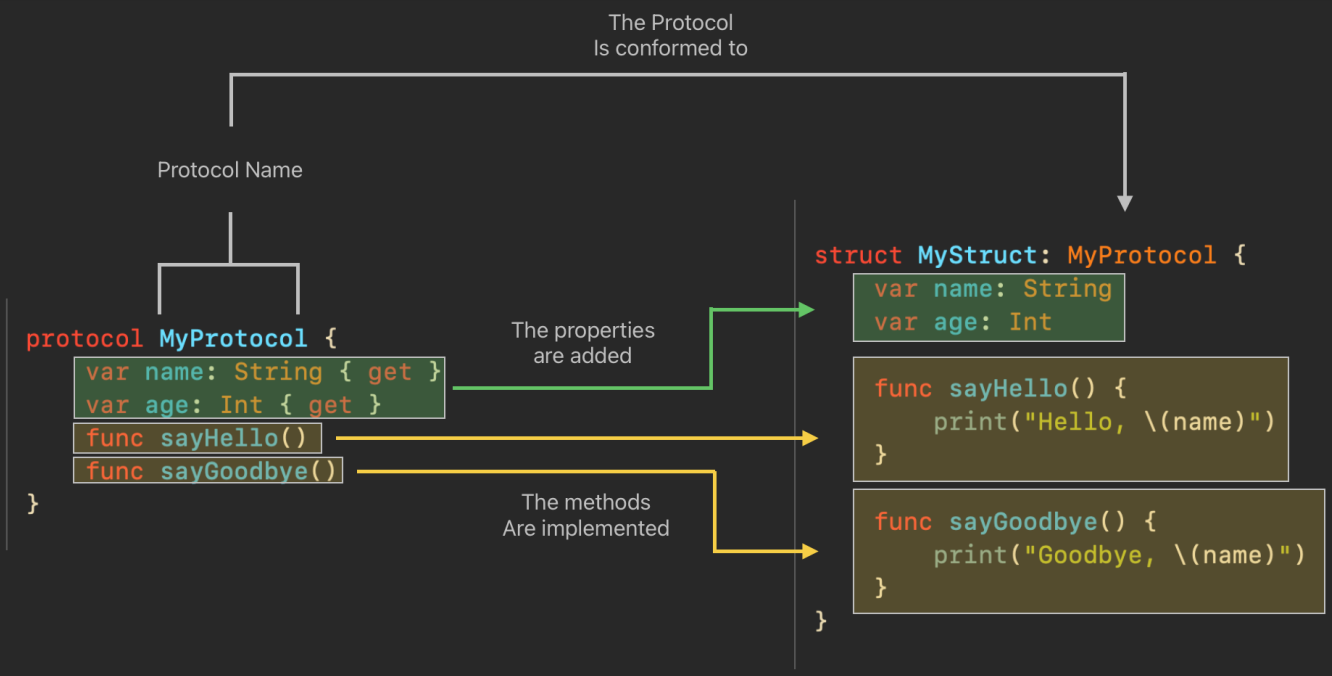
You write the keyword protocol, give it a name, and—within curly braces—define its properties and methods.



- Note that only the method signature is written, not its implementation.

How does a conforming type (structure, enumeration, or class) conform to a protocol?

After the name of the type, you write a colon (:), followed by the name of the protocol to conform to (in uppercase, as it refers to a data type).



Finally, this is how our MyStruct structure looks when conforming to the MyProtocol protocol.

```
struct MyStruct: MyProtocol {
    var name: String
    var age: Int

    func sayHello() {
        print("Hello, \(name)")
    }

    func sayGoodbye() {
        print("Goodbye, \(name)")
    }
}
```

What we've learned so far is what protocols are, why they're important, and how a structure, class, or enumeration conforms to them and implements their properties and methods.
In future posts, we'll continue to explore new questions:

- When are protocols more effective than class inheritance?
- What situations can be resolved only with protocols and not in other ways?
- Do Swift protocols have an equivalent in languages like Java, JavaScript, C++, .NET, or Python?

Entonces, hasta la próxima!