# Livro do Educador

## Desenvolvimento de Aplicativos Móveis

### iOS

# Unidade 5

## Aula 1 – Protocolos

Os protocolos estão bem presentes nas linhas de programação dos aplicativos iOS, como as interfaces em Android, e são responsáveis por definir métodos, propriedades e outros que em conjunto, definem uma funcionalidade ou parte dela.

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Compreender a definição e uso de Protocolos;
* Conhecer as propriedades gettable e settable;
* Definir restrições de métodos, propriedades e initializers com os protocolos para objetos;
* Utilizar extensões para acrescentar funcionalidades a um objeto e também oferecer conformidade com protocolos adotados;

Realizar checagem de conformidade com os Protocolos.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 40 minutos de aula expositiva;
* 10 minutos para tirar as dúvidas dos alunos;
* 40 minutos para desenvolver as atividades propostas para a turma e apresentação da TDP.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Sintaxe do Protocol;
* Requisitos de conformidade;
* Protocolos como tipos de dados;
* Extensions;
* Checando conformidade do protocolo.

### PONTOS IMPORTANTES

É importante que o aluno se atente a essa aula pois será apresentados bastante exemplos em códigos, que são de crucial entendimento para o decorrer da aula.

### Sintaxe do Protocol

Os protocolos podem ser adotados por classes, estruturas ou enumerações para prover uma implementação real dos requisitos.

Educador, explique que para definir um protocolo, utilizamos a palavra reservada protocol seguida de um nome para o protocolo à escolha, SomeProtocol e entre as chaves { } é feita sua definição.

Também para que um tipo adote um protocolo, utilizar a palavra reservada para o tipo e em seguida o seu nome (class SomeClass), colocar “:” e definir o protocolo a ser adotado e caso ele seja herdado de outra classe, inserir logo a pós os : o nome da super classe e em seguida o nome do protocolo, separado por virgula (class SomeClass: SomeSuperclass, FirstProtocol ). È possível adotar mais de um protocolo para cada tipo, bastando apenas, inserir uma vírgula e o nome de outro protocolo após o último que foi adicionado.

### Requisitos de conformidade

Em teoria uma classe está em conformidade com um dado protocolo quando implementa todos os seus métodos não opcionais. O protocolo é uma espécie de acordo que diz quais métodos uma classe deve implementar para entrar em conformidade para prover uma propriedade, mas o Protocolo não especifica se ela deve ser armazenada ou computada. Ele pode especificar seu requisito como somente **gettable** ou se é **gettable e settable**.

#### Propriedades

Educador, resumidos nas tabelas a seguir, temos as formas de implementação de algumas definições e restrições para os protocolos, tome como forma de acesso rápido às informações:

|  |  |
| --- | --- |
| **Propriedade** | **Requisito satisfeito** |
| gettable e settable | Por propriedade que não seja stored ou ready-only. |
| gettable | Por qualquer propriedade. |

Para declarar estes requisitos temos:

|  |  |
| --- | --- |
| **Implementação** | **Definição** |
| protocol SomeProtocol {    var mustBeSettable: Int { get set }    var doesNotNeedToBeSettable: Int { get }  } | Os requisitos das propriedades devem ser declarados como variáveis. |
| protocol SomeProtocol {    var mustBeSettable: Int { get set }    var doesNotNeedToBeSettable: Int { get }  } | Para indicar propriedades gettables e settables. |
| protocol SomeProtocol {    var mustBeSettable: Int { get set }    var doesNotNeedToBeSettable: Int { get }  } | Para indicar propriedades apenas gettables. |
| protocol AnotherProtocol {    static var someTypeProperty: Int { get set }  } | Indica que o tipo personalizado que adota o protocolo deverá ter uma variável estática e não o protocolo. |

Educador, utilizar o exemplo do protocolo **FullyNamed**, que em base, estabelece que qualquer um que conforme com **FullyNamed** (com nome completado – totalmente nomeado) deva ter uma propriedade de instancia (atributo) **gettable** chamada **fullName** (nome completo) do tipo **String**. Dê também o exemplo da estrutura **Pearson**, que adota o protocolo FullyNamed. Ela terá uma única propriedade, a fullName, que conincide com a requisição do protocolo adotado, sendo assim o protocolo é cumprido.

Explique como que com a palavra **return**, fazemos com que a propriedade computada **fullName** seja apenas **gettable** e usando as palavras reservadas get e set para termos a propriedade fullName como gettable e settable. De as explicações finais e vamos aos métodos.

#### Métodos

Para escrever métodos de instância ou de tipo normal exigidos pelo protocolo, devemos escrevê-los como parte da definição do protocolo, sem chaves ou um corpo do método.

Educador, aqui temos o protocolo **RandonNumberGeneretor** , a sua restrição é que o tipo tenha um método **random()**  e que ele retorne um valor **Double**. Em seguida é mostrado um exemplo de uma classe que adota o protocolo **RandonNumberGeneretor**, inicia algumas variáveis com alguns valores e define um método **random()** que retorna um valor do tipo **Double**, então a restrição do protocolo é satisfeita e no termino desta classe os valores são apresentados.

#### Initializers

Outra conformação que os protocolos podem exigir é que inicializadores sejam implementados. E são definidos como parte da definição do protocolo, sem chaves ou um corpo inicializador.

##### 1.2.3.1. Implementando um initializer requerido pelo protocolo

Utilizar o modificador required para a implementação fornecida seja explicita ou herdada do inicializador em todas as subclasses das classes conformes. Utilizando ambos os modificadores required e override em **uma subclasse**, podemos anular um **inicializador** adotado por **uma superclasse** e implementar um **requisito inicializador** correspondente a **um protocolo.**

### Protocolos como tipos de dados

Um protocolo pode ser utilizado para dedeterminar um tipo de dado a ser utilizado, e podendo ser um tipo, é possível utilizar os protocolos como um tipo de parâmetro, tipo de retorno em uma função, tipo de constante, variável ou propriedade, tipo de arraiy items, dicionários ou outro tipo de recipiente. Para este uso aconselha-se o uso de nomes de protocolos iniciados com letra maiúscula para se assemelhar com a nomenclatura dos tipos.

Educador explique com o exemplo da classe **Dice** que utiliza o protocolo **RandomNumberGenerator** como **tipo** da propriedade (e também do parâmetro do inicializador) **generator** e por isso é possível atribuir a generator uma instancia de qualquer tipo desde esteja em conformidade com RandomNumberGenerator ao iniciar uma nova instância da classe Dice, já que generator também é parâmetro do inicializador e tem como tipo este protocolo.

A classe roll possui o método roll, que utiliza o método random() para atribuir um valor aleatório entre 0.0 e 1.0 à propriedade genetator. Como ele recebe o tipo RandomNumberGenerator, é garantido que a propriedade tenha o método random() para ser chamado. Explique o último exemplo para mostrar esta propriedade dos protocolos.

### Extensions

Educador, explique que as extensões podem adicionar novas funcionalidades a uma classe, estrutura, enumeração ou tipo de protocolo existente e também é uma forma de organizar o código. Dê alguns exemplos destas funcionalidades e vejamos a sintaxe da Extension.

#### Sintaxe da Extension

Extensions são declaradas da seguinte forma.

|  |  |
| --- | --- |
| **Implementação** | **Definição** |
| extension SomeType: SomeProtocol, AnotherProtocol {    // A implementação dos requisitos dos protocolos vão aqui  } | Utiliza-se a palavra reservada (extension) seguida do nome (SomeType) para definir uma extensão. |
| extension SomeType: SomeProtocol, AnotherProtocol {    // A implementação dos requisitos dos protocolos vão aqui  } | Uma extensão que estende um tipo existente pode adotar um ou mais protocolos. |

#### 1.4.2. Conformidade de protocolos com Extensions

Extensões podem adotar e estar conforme com um novo tipo de protocolo, e como ela adiciona novas propriedades, ela á capaz de adicionar todos os requisitos exigidos para conformidade de um protocolo.

Mostre o novo protocolo e como a extensão o adota. É como se a classe Dice tivesse fornecido na sua implementação original e agora, qualquer instancia de Dice pode ser tratada como TextRepresentable. Sendo assim, a extensão adiciona uma propriedade a classe Dice que faz com que ela entre em conformidade com o protocolo. Não confundir este tipo de extensão com uma relação de herança como vimos nas outras linguagens.

### 1.5. Checando conformidade com protocolo

Para checar e converter protocolos utiliza-se os operadores **is** e **as**. Como na verificação e conversão de tipos, temos:

|  |  |
| --- | --- |
| **Operador** | **Definição** |
| **is** | Retorna true se uma instancia está em conformidade com o protocolo e false caso contrário. |
| **as?** | Se a instancia estiver conforme o protocolo, retorna um valor opcional do tipo do protocolo, caso contrário, retorna **nil**. |
| **as!** | Força a conversão para o tipo do protocolo. |

No exemplo é dado o protocolo HasArea, com sua propriedade area do tipo Double, gettable.

As duas classes citadas adotam o protocolo e precisam ter a propriedade area do tipo Double para satisfazer o protocolo. Verificando o código, as classes estão conformes, porque também são gettable.

Já a classe Animal não se conforma com o protocolo. Depois disto é criado um array listando as classes e iterando seus objetos. Em seguida é criado uma forma de verificar a conformidade com o protocolo, com um repetidor e um condicional. Ele usa o operador **as?** que retorna true se a instancia estiver conforme o protocolo e imprimirá certa mensagem, caso não retorne true (else) irá imprimir outra mensagem.

### 1.6. Resumo

Nesta aula o aluno se aprofundou um pouco mais no assunto de protocolos, aprendeu os diferentes tipos de conformidades como propriedades (get e set) e métodos específicos, inicializadores. Viram também que protocolos podem ser usados como tipos de dados e que é possível que extensões adicionem mais funcionalidades em objetos, incluindo conformidades com protocolos adotados. Por último o aluno aprendeu como é feita a checagem das conformidades dos protocolos, utilizando os operadores is e as.

## Aula 2

### 2. Delegates e gerenciamento de dependências

Vamos conhecer o nosso primeiro padrão de projeto, o delegate. Um mecanismo muito importante no qual um objeto age em nome de ou em coordenação com outro objeto. Também vamos aprender a utilizar bibliotecas de terceiros, poupando muito de nosso tempo de trabalho.

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Vizualizar as vantagens de uso de bibliotecas e gerenciadores de dependência;
* Definir o significado de gerenciadores de dependência;
* Instalar e configurar o CocoaPods e a biblioteca Firebase.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 40 minutos de aula expositiva;
* 10 minutos para tirar as dúvidas dos alunos;
* 40 minutos para desenvolver as atividades propostas para a turma e apresentação da TDP.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Padrão de projeto Delegate;
* CocoaPods.

### PONTOS IMPORTANTES

É interessante traçar paralelos entre as aulas de Android e iOS e ressaltar semelhanças entre o uso de bibliotecas dos dois. Também se atente a ressaltar as vatagens de utilizá-las e enfatizar ao aluno que o uso de bibliotecas de terceiros não envolve nada parecido com cópia, plágio ou pirataria. O uso delas deve ser visto como algo positivo para a produtividade.

### 2.1. Padrão de projeto Delegate

Educador defina, em iOS, o que delegação é um mecanismo no qual um objeto age em nome de, ou em coordenação com, outro objeto.

Dê o exemplo do método em que definimos a quantidade de linhas em cada seção numa Table Views, que não é capaz de realizar este cálculo de linhas, mas essa tarefa pode ser realizada pelo protocolo **UITableViewDelegate**, tornando a classe UITableView independente dos dados que ela exibe, ela só terá o trabalho de exibi-lo e não precisará buscar informação que não tem, ela passa essa responsabilidade para o delegate. Explique, por que é vantajoso utilizar este padrão e cite algumas classes que ele é empregado.

#### 2.1.1. Delegates e DataSources

Educador, explique a diferença destes dois objetos criados no padrão Delegate. O primeiro cuida da manipulação de interface quanto o segundo usa o padrão para obter dados para a aplicação.

### 2.2. CocoaPods

Educador, iniciando o assunto sobre gerenciadores de dependências, apresente ao aluno o CocoaPods, oferecendo a definição dada pelo time desenvolvedor.

Dê uma explicação básica sobre o que é gerenciamento de dependências em iOS. As dependências são partes do projeto que está sendo resolvida por métodos de terceiros (frameworks ou bibliotecas de terceiros) e não está necessariamente no local do seu projeto. É chamada dependência, pois sem ela o seu projeto simplesmente não funciona. O gerenciador de dependências é responsável por instalar estes códigos de terceiros.

#### 2.2.1. Porque utilizar bibliotecas de terceiros

Educador, explique que uma das vantagens é que, ao invés de ficarmos escrevendo incontáveis linhas de código que já foi escrita por alguém, e convenientemente funcionam muito bem, podemos utilizar estes códigos já escritos e gastarmos nosso tempo com outras coisas, como dar um polimento maior para nosso app.

Além disso, ao usarmos bibliotecas podemos mantê-las atualizadas nas suas versões mais recentes com muito mais facilidade do que atualizar nossos códigos manualmente. Além de ser muito mais fácil encontrar bibliotecas em um site que centraliza todas elas disponíveis como o CocoaPods. Entre outras vantagens

O CocoaPods:

* instala as bibliotecas no projeto;
* resolve a questão das subdependências entre bibliotecas;
* auxilia na busca de novas bibliotecas;
* configura o ambiente de desenvolvimento apropriado para o uso destas bibliotecas.

#### 2.2.2. Instalando o CocoaPods

Para instalar o CocoaPods utilizamos o seguinte método:

* No terminal do Mac, digitar o comando como no passo 1 e inserir a senha de usuário do Mac (se necessário, forneça a senha da escola). A instalação começará. Executar o segundo comoandono terminal, como no passo 3 para concluir a instalação.

#### 2.2.3. Configurando nosso projeto

Para que o projeto receba as bibliotecas:

* Abrir o terminal e inserir o comando como no passo 1 e executar o comando, no passo 2. O arquivo Podfile será criado.
* Abrir o Podfile com o comando mostrado no passo 3 e verificar seu conteúdo que será semelhante ao código mostrado no passo 4.
* Para utilizar a biblioteca Firebase:
* Inserir no bloco target a instrução mostrada no passo 1 e verificar o resultado no terminal.
* Verificar que o arquivo **MusicProject.xcworkspace** foi criado, que é o que deveremos abrir daqui para frente. Não usar mais o arquivo o arquivo.xcodeproj.

### 2.3. Resumo

O aluno aprendeu nessa aula como é possível economizar tempo utilizando bibliotecas de terceiros. Conheceu o CocoaPod e como ele funciona para gerenciar dependências e também ficou a par de algumas de suas vantagens de uso e por fim aprendeu a instalar e configurar bibliotecas.

## Aula 3

### 3. Closures

Para criarmos métodos e passarmos ele via parâmetro em alguns métodos ou atribuí-lo em alguma variável ou constante, utilizamos os **closures.**

### CARGA HORÁRIA

Conforme o plano de aula, esta aula terá duração de 1h30 e deverá ser conduzida de acordo com as orientações pedagógicas.

### OBJETIVO DA AULA

Ao final da aula, você deverá garantir que o aluno tenha subsídios para

* Fazer uso e ter os closures como simplificadores e organizadores;
* Simplificar closures o máximo que for necessário;
* Utilizar método de ordenação.

### ORIENTAÇÕES PEDAGÓGICAS

Para atender os objetivos de aprendizagem, você deverá conduzir o processo de ensino considerando a organização didática apresentada a seguir:

* 40 minutos de aula expositiva;
* 10 minutos para tirar as dúvidas dos alunos;
* 40 minutos para desenvolver as atividades propostas para a turma e apresentação da TDP.

### TÓPICOS DE ESTUDO

Todos os tópicos a seguir, conforme livro do aluno, devem ser trabalhados de forma dinâmica, criativa, com embasamento teórico e prático voltado ao mercado de trabalho.

* Expressões closure;
* Sintaxe de um Closure;
* Trailling Closure.

### PONTOS IMPORTANTES

Professor, esta aula é bem acessível ao aluno, mas cuidado para que ele não tenha um conceito de que podemos usar sempre os closures da maneira mais simplificada possível.

### 3.1. Expressões Closure

Educador, para explicar o que são closures, comece explicando que o aninhamento de funções é possível em Swift, mas o problema é que o código fique complexo e pouco legível. Ao invés de fazer estes aninhamentos utiliza-se os **closures**, que **otimizam a escrita** de certos algoritmos sem diminuir sua clareza ou intenção, isso através de blocos de códigos escritos inline em uma sintaxe breve e focada. Vejamos exemplos mais a frente.

#### 3.1.1. O método de ordenação

O método **sorted(by:)** é um método de ordenação que classifica um array de valores de um tipo específico, com base numa saída ordenada por um closure. No caso do exemplo, é um array de Strings de nomes que queremos ordenar em ordem alfabética reversa. Então definimos uma função que retorna um valor true se uma String 1 for maior que uma String 2, significa um caractere aparece antes no alfabeto do que outro, então esse segundo é maior que o primeiro. E colocamos esta função na definição do método de ordenação, assim uma variável receberá um array de strings em ordem alfabética reversa.

### 3.2. Sintaxe de um closure

Dado o exemplo anterior, agora vamos substitur a função por um closure, onde os parâmetros e retorno (que antes foi a função backward) deste closure inline, agora escritos dentro das chaves. A palavra reservada **in** indica que a definição dos parâmetros e tipo de retorno terminou e dali para frente tem se o corpo do closure.

Com este último exemplo vemos que o closure é capaz de simplificar a quantidade de código que no caso uma função poderia requerer.

#### 3.2.1. Inferindo tipo pelo contexto

Educador, ainda utilizando o exemplo do método **sorted(by:)** vemos que não é necessário definir o tipo de retorno, pois ele é chamado de um array de Strings, então o tipo de parâmetros e retorno do closure, não precisam ser escritos em seu argumento pois são definidos pelo contexto, logo o argumento deste método será do tipo **(String, String) -> Bool**.

#### 3.2.2. Retornos implícitos em closures com uma única expressão

Educador, a palavra reservada **in** pode ser omitida da declaração do closure, como visto no exemplo. No caso deste método de ordenação, está claro que o closure deve devolver um valor **Bool** para o método, pois não possui expressão ambígua. Ainda é possível reduzir o closure ainda mais.

#### 3.2.3. Nomes de argumentos taquigráficos

Educador, é possível ainda omitir a lista de argumentos e como consequência a palavra reservada **in**, utilizando argumentos taquigráficos, que são abreviações do nome dos argumentos dos métodos. Ele utiliza $0, $1, ..., para representar os argumentos do método.

Ainda há mais uma simplificação.

#### 3.2.4. Métodos de operador

Educador, explique o comportamento dos operadores de comparação como métodos. Operações de comparação devem emitir um sinal **true** se a comparação for verdadeira e logicamente, uma comparação é feita sempre entre dois valores, um de cada lado do operador. No caso do >, o Swift já prevê que os valores estarão do lado esquerdo e direito do comparador, sendo desnecessário, para este método de ordenação declarar que há um valor dos dois lados do comparador.

Assim concluímos que o método **sorted(by:)** necessita apenas do operador > como parâmetro do método.

Mas mesmo tendo todas essas formas de simplificação é sempre melhor fazer a declaração completa para evitar erros.

### 3.3. Trailling Closures

Educador, Trailling Closurer é um closure que pode ser arrastado para fora do método e reaproveita-lo em lugares diferentes. É escrito depois dos parênteses da chamada da função e não é necessário escrever o rótulo do argumento para a parte do closure na chamada da função. Compare uma chamada de closure convencional e uma de trailling closure. Notar que na chamada convencional, o closure é o único argumento do método, então é possível fornecer essa função como trailling closure, sem mesmo os “( )” após o nome do método. A função **sorted(by:)**  se enquadra neste caso. Mostre os exemplos e partamos para as aplicações.

#### 3.3.1. Preparando a camada de serviço

Como vimos no começo da aula, não é fácil trabalhar com conceitos de Networking no iOS. Para o aluno refrescar a memória, relembre um pouco dos conceitos vistos em Android sobre REST e HTTP e como usamos a biblioteca Ion para fazer requisições GET de uma determinada URL, que vem no formato JSON. Relembre também a maneira assíncrona que estas requisições eram trabalhadas e como podemos reproduzir isso com chamadas assíncronas utilizando os closures em Swift.

### 3.4. Resumo

Nesta aula, o aluno aprendeu sobre os **closures** e seu grande potencial para organizar e limpar o código, evitando funções desnecessárias e também simplificando ao máximo sua declaração.