

INTRODUÇÃO AO ORIENTAÇÃO A OBJETOS

Orientação a Objetos é um paradigma de programação que parte do princípio que todo nosso programa é dividido em partes.

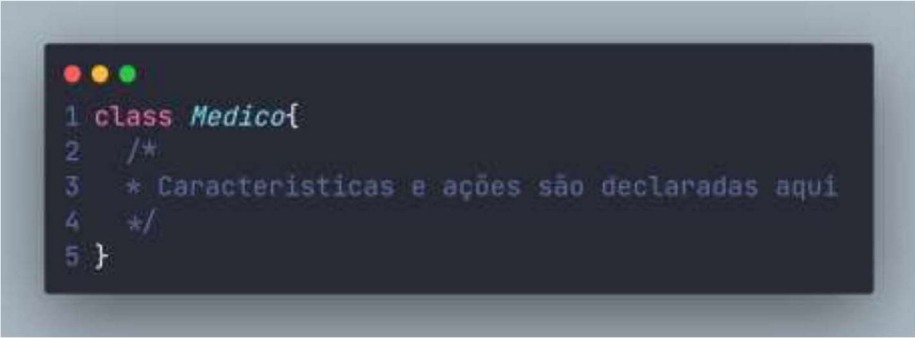
Essas partes são chamadas de objetos. Sendo que esses objetos são as entidades que representam o sistema.

Classes

Um dos conceitos fundamentais na orientação a objetos é a classe. Uma classe é uma estrutura que define as características de um elemento do programa, ou seja, as partes que compõem o sistema. Em um exemplo simples, caso estejamos criando um sistema para uma clínica médica teremos elementos como: **médicos**, **enfermeiros**, **pacientes**, **atendentes** e etc. Todos esses são elementos do sistema e tem suas características e ações dentro de nosso sistema. Sendo assim, utilizamos a classe para definir tais características e ações.

No Dart, assim como em outras linguagens de programação, utilizamos um arquivo para declarar uma classe. E dentro dessa estrutura de classe, definimos todas as características e ações pertinentes a esse elemento de nosso sistema.

Vejamos agora um exemplo de uma estrutura de classe Dart.

A imagem mostra um editor de código com um fundo escuro. No topo, há três botões de janela (vermelho, amarelo, verde). O código é escrito em uma fonte monoespaçada com coloração de sintaxe: a palavra-chave 'class' é em rosa, 'Medico' em verde, e os caracteres de abertura e fechamento de bloco de comentário '/*' e '*/' são em amarelo. O código consiste em cinco linhas numeradas de 1 a 5.

```
1 class Medico{
2     /*
3     * Características e ações são declaradas aqui
4     */
5 }
```

No exemplo apresentado vamos um exemplo de uma classe Dart, onde utilizamos a palavra-chave “**class**” para declarar que a estrutura representa uma classe Dart. Logo após a declaração da classe definimos o nome da classe, que em nosso exemplo é a “**Medico**”.

Visibilidade

Assim como em outras linguagens de programação Orientadas a Objetos, o Dart possui uma característica peculiar que caracteriza a estrutura de uma classe, a

visibilidade dos elementos que compõem a classe, ou seja, suas propriedades e ações.

No Dart, assim como em outras linguagens de programação que implementam o paradigma “Orientação a Objeto”, encontramos dois tipos distintos de visibilidade, são elas:

Public: Como já visto em nosso exemplo, define que um determinada propriedade ou método pode ser utilizado por qualquer parte de nosso sistema, sem qualquer restrição de acesso ou manipulação.

Private: Essa visibilidade é o oposto da public, sendo que só permite o acesso e/ou manipulação por elementos pertencentes a própria classe em que tal propriedade ou método se encontra. Sendo assim, somente um método da própria classe em que a propriedade foi declarada pode acessar e modificar o valor armazenado na mesma. No Dart qualquer objeto que possuir um sinal de underline () em frente ao seu nome é considerado como privado.

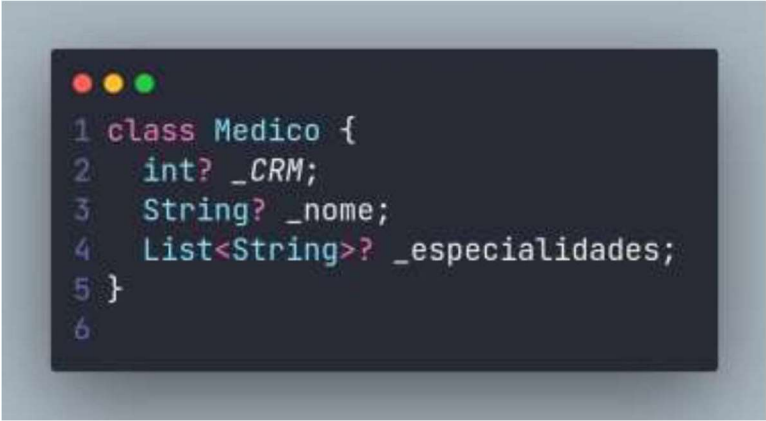
Propriedades da classe

Como já vimos anteriormente, uma classe é composta por características (Propriedades) e ações (Métodos).

As propriedades de uma classe nada mais são que as características que determinam como um elemento do sistema é ou com o que se parece. Se pararmos para pensar na premissa da Orientação a Objetos, encontraremos que seu intuito é tornar o sistema o mais parecido com o mundo real. Ou seja, tornar nossos códigos mais parecidos com nosso cotidiano. Sendo assim podemos tomar como regra que todas as classes representam elementos reais de nossas vidas, como pessoas, animais, profissões e etc. E como bem sabemos, todos esses têm características que os definem, uma pessoa tem nome, peso, altura e etc. Um animal tem uma espécie, uma cor predominante, um som característico e por aí vai.

Bom para não acabarmos por nos perder em nosso assunto, vamos focar na declaração de propriedades de uma classe. Para tal devemos utilizar a mesma sintaxe que utilizamos para a declaração de nossas variáveis.

Também podemos optar por qualquer visibilidade que desejarmos. Mas é uma convenção que as propriedades de uma classe sejam privadas. Mas nada lhe impedirá de declará-las como públicas.



```
1 class Medico {
2     int? _CRM;
3     String? _nome;
4     List<String>? _especialidades;
5 }
6
```

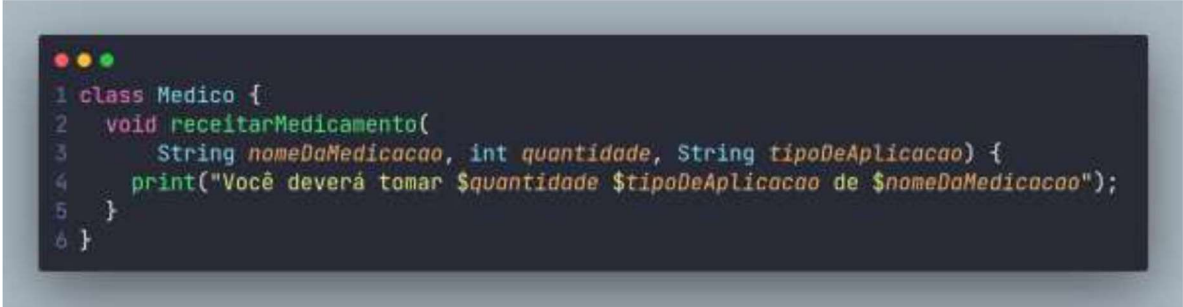
Como podemos ver em nosso exemplo, nosso médico possui três propriedades: um nome, o seu CRM (Registro no conselho regional de medicina) e sua especialidade clínica. Como podemos ver no comentário apresentado, podemos declarar quantas propriedades forem necessárias para definirmos quais são as características de um médico.

Métodos da classe

Outra característica importante de uma classe são as suas ações, também chamados de métodos de classe ou simplesmente de métodos. Uma classe pode possuir um ou vários métodos, ou mesmo não os possuir.

Um método de classe tem como objetivo definir ações comuns de um elemento do sistema, como por exemplo realizar um determinado cálculo ou mesmo inicializar uma propriedade da mesma.

Um método de classe não se difere muito de um método, pois ambos tem uma assinatura, ou seja, um nome e seus parâmetros. O que diferencia um método de classe de uma simples função é a adição de uma visibilidade ao mesmo, normalmente pública, mas não é uma regra.



```
1 class Medico {
2     void receitarMedicamento(
3         String nomeDaMedicacao, int quantidade, String tipoDeAplicacao) {
4         print("Você deverá tomar $quantidade $tipoDeAplicacao de $nomeDaMedicacao");
5     }
6 }
```

Encapsulamento e métodos acessores

Outro aspecto importante da Orientação a Objetos é o encapsulamento, ou seja, a possibilidade de “encapsular” as propriedades de uma classe, assim dando um controle maior a classe sobre elas. Dessa forma, podemos declarar quem e sob quais circunstâncias uma determinada parte do sistema poderá acessar tais propriedades.

Talvez você já deva estar se lembrando de quem estamos falando, são os nossos tão conhecidos métodos acessores GET e SET.

Esses métodos são criados para dar acesso controlado a uma determinada propriedade privada presente em uma classe. Com eles podemos criar regras para o acesso a essas propriedades, assim como negar acesso às mesmas. Vejamos um exemplo prático da aplicação deste conceito.



```
1 class Medico {
2     int? _CRM;
3     String? _nome;
4     List<String>? _especialidades;
5
6     // Métodos Get
7     int? get CRM => this._CRM;
8     get nome => this._nome;
9     get especialidades => this._especialidades;
10    // Métodos Set
11    set CRM(int? value) => this._CRM = value;
12    set nome(value) => this._nome = value;
13    set especialidades(value) => this._especialidades = value;
14 }
15
```