**Aula IX e X - DevAppII**

**Introdução a Orientação a Objeto**

**Conceitos sobre a Orientação a Objeto**

**Classes**

**Encapsulamento**

**Métodos**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

***Fontes:***

<https://flutter.dev/>

<https://material.io/design>

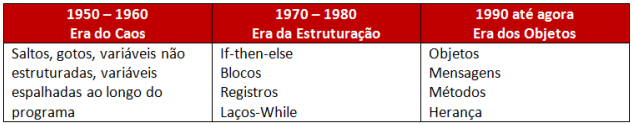
<https://www.flutterparainiciantes.com.br/basico/stateless-stateful>

<https://www.flutterbrasil.com/31-orientacao-a-objetos-em-dart-classes-atributos-e-objetos>

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Programação Orientada a Objeto**

Até o início da década de 70, o computador era utilizado somente por grandes empresas. Neste período, com a queda de preço dos computadores e a consequente proliferação de uso destes, cresceu a demanda por software. As técnicas de desenvolvimento de software utilizadas até então não eram suficientes para contornar problemas existentes no desenvolvimento de sistemas, principalmente quando desenvolvidos em grande escala, como então se exigia. Na verdade, pouco se possuía de técnicas que estivessem realmente sendo aplicadas.



**Paradigma Imperativo - Estruturado**

Este paradigma é o primeiro paradigma a surgir e até hoje dominante. É um paradigma baseado na arquitetura Von Neumann. Segue o conceito de estruturação, onde tudo possui uma sequência passo a passo a ser seguida, modificando os dados a fim de chegar ao resultado esperado.

**Exemplos: C, Pascal, Basic**.

Foi neste contexto que surgiu a programação estruturada, seguida pelo conceito de desenvolvimento estruturado de sistemas. Esta metodologia tentava oferecer soluções para os problemas ligados ao desenvolvimento de sistemas, ao pregar a aplicação dos seguintes princípios:

* **Princípio da abstração:** para resolver um problema, o analista deveria analisá-lo separadamente dos demais aspectos, ou seja, abstrair os detalhes;
* **Princípio da formalidade:** o analista deveria seguir um caminho rigoroso e metódico para solucionar um problema;
* **Princípio de “dividir para conquistar”**: o analista deveria dividir o problema em partes menores, independentes e com possibilidade de serem mais simples de entender e solucionar;
* **Princípio da disposição hierárquica:** o analista deveria organizar os componentes da solução do problema na forma de uma árvore com estrutura hierárquica. O sistema seria entendido e construído nível a nível, onde cada novo nível acrescentaria mais detalhes.

**Na programação orientada a Objetos**

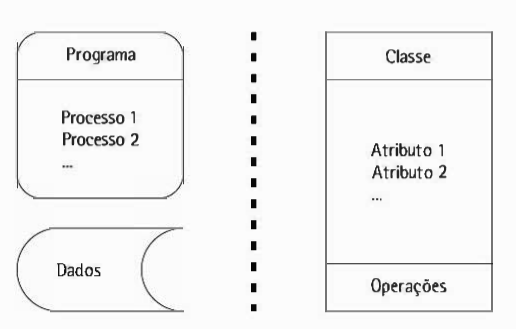
Os objetos são a chave para a compreensão da tecnologia orientada a objeto. Olhe em volta agora e você vai encontrar muitos exemplos de objetos do mundo real: seu cachorro, sua mesa, sua televisão. Os objetos do mundo real partilham duas características: eles possuem estado e comportamento. Os cães têm estado (nome, cor, raça) e comportamento (latidos, cheirando, abanando o rabo). Bicicletas também

A programação orientada a objetos **representa uma mudança no enfoque da programação,** na forma como os sistemas eram vistos até então. Representa uma quebra de paradigma, revolucionando todos os conceitos de projeto e desenvolvimento de sistemas existentes anteriormente.

O enfoque tradicional para o desenvolvimento de sistemas e, por consequência, para a programação, **baseia-se no conceito de que um sistema é um conjunto de programas inter-relacionados que atuam sobre um determinado conjunto de dados que se deseja manipular de alguma forma para obter os resultados desejados.**

**O enfoque da modelagem de sistemas por objetos procura enxergar o mundo como um conjunto de objetos que interagem entre si e apresentam características e comportamento próprios representados por seus atributos e suas operações.** Os atributos estão relacionados aos dados, e as operações, aos processos que um objeto executa. Assim, supondo que se deseje desenvolver um sistema de controle de estoque para uma empresa, procura-se identificar os objetos relacionados ao sistema, como os produtos, os pedidos de compra, os recibos, as pessoas etc., conforme está detalhado a seguir.

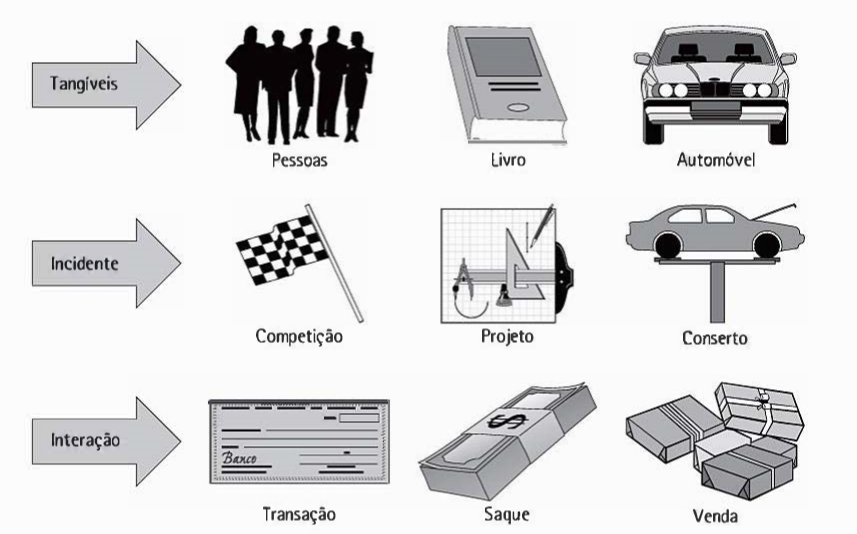
Pode-se dizer que é possível modelar, por meio da orientação a objetos, um setor, um departamento e até uma empresa inteira. Esse enfoque justifica-se, de forma resumida, pelo fato de que os objetos existem na natureza muito antes de haver qualquer tipo de negócio envolvido ou qualquer tipo de sistema para controlá-los. Equipamentos, pessoas, materiais, produtos, peças, ferramentas, combustíveis etc. existem por si sós e possuem características próprias determinadas pelos seus atributos (nome, tamanho, cor, peso) e um determinado comportamento no mundo real relacionado aos processos que eles sofrem.

Programação Tradicional/Estruturada Programação Orientada a Objeto

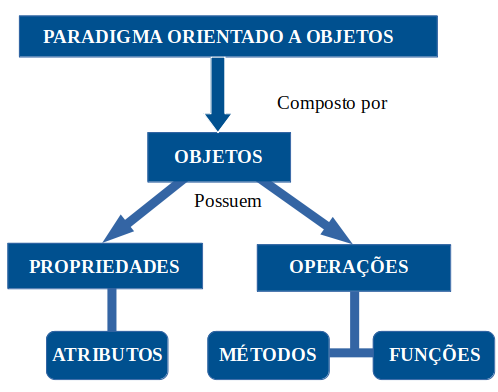
**Os 4 pilares da Programação Orientada a Objeto (POO)**



**0 que é um Objeto**

**Um dos primeiros conceitos básicos da orientação a objetos é o do próprio objeto.** Um objeto é uma extensão do conceito de objeto do mundo real, em que se podem ter coisas tangíveis, um incidente (evento ou ocorrência) ou uma interação (transação ou contrato). 

Fonte da Imagem do Livro Lógica de Programação - Sandra Puga



**Classes no Dart/Flutter**

É um modelo ou um plano para um objeto, exemplificando seria uma planta para se construir uma casa. É um arquivo onde você concentra todas as características daquele objeto. No exemplo a seguir iremos criar uma classe chamada pessoa. Veja como se estrutura uma classe class NomeDaClasse {} :

**class Pessoa {**

**}**

**ESTRUTURA DAS CLASSES**

**Atributos**

Dentre dessa classe pessoa temos os atributos desta classe: nome, idade e altura. Esses atributos são as informações que o nosso objeto irá armazenar.

|  |
| --- |
| **class Pessoa {**  **// atributos**  **String nome;**  **int idade;**  **double altura;**  **}** |

**Métodos (Funções)**

Agora que o objeto já está definido vamos ver o que uma pessoa é capaz de fazer, para definir suas ações as fazemos por meio de funções void que chamamos, quando estão dentro das classes, de métodos:

|  |
| --- |
| **class Pessoa {**  **// atributos**  **String nome;**  **int idade;**  **double altura;**  **// métodos**  **void dormir(){**  **print("$nome está dormindo");**  **}**  **}** |

**OBS**. Nas funções, nós aprendemos que void é uma função que não retorna nada para quem a requisitar.

**Objetos**

Podemos criar objetos por meio desta classe, ou seja, declará-los ou instanciar nossas classes, portanto, hipoteticamente falando seria "criar as pessoas". Para instanciá-las, fazemos da seguinte forma:

**Pessoa pessoa1 = new Pessoa();**

Aplicando...

|  |
| --- |
| **class Pessoa {**  **String nome;**  **int idade;**  **double altura;**  **void dormir(){**  **print("$nome está dormindo");**  **}**  **}**  **void main(){**  **// criando um objeto, pode começar com var, se quiser**  **// aqui instanciamos nossa pessoa**  **Pessoa pessoa1 = new Pessoa();**  **// em Dart o new é opcional, pode-se colocar ou não, na dúvida coloca para ficar claro que está criando uma nova pessoa**  **}** |

Primeiro criamos o tipo, ou seja, é do tipo "Pessoa", depois o nome "pessoa1", depois recebe um novo objeto do tipo **Pessoa();** que seria um instanciamento de um objeto (é exatamente igual a declarar uma variável **String algumacoisa = "")**.

**Resumindo: estamos criando a pessoa1. Se dermos um print nessa pessoa1, irá aparecer "Instance of 'Pessoa'", ou seja, quer dizer que agora o pessoa1 é uma instância da classe Pessoa.**

Desta forma, já que criamos uma pessoa (pessoa1) vamos declarar os dados desta pessoa.

void main(){

Pessoa pessoa1 = new Pessoa();

// acessando os atributos

pessoa1.nome = "Ana";

pessoa1.altura = 1.70;

pessoa1.idade = 30;

print(pessoa1.nome);

}

Coloca-se o **"."** para acessar tal atributo. Pode-se, portanto, criar outra pessoa (pessoa2) que será criada da mesma fonte, mas com dados diferentes.

**Chamando um método**

Para chamar um método utilizamos também o "." e depois o nome do método, conforme abaixo:

|  |
| --- |
| **class Pessoa {**  **String nome;**  **int idade;**  **double altura;**  **void dormir(){**  **print("$nome está dormindo");**  **}**  **void aniver(){**  **idade++;**  **}**  **}**  **void main(){**  **Pessoa pessoa1 = new Pessoa();**  **pessoa1.nome = "Diego";**  **pessoa1.altura = 1.70;**  **pessoa1.idade = 30;**  **print(pessoa1.nome);**  **pessoa1.aniver();**  **print(pessoa1.idade);**  **}** |

No script acima irá retornar a idade conforme a função **aniver();,** ou seja irá retornar o número 31 (30+1).