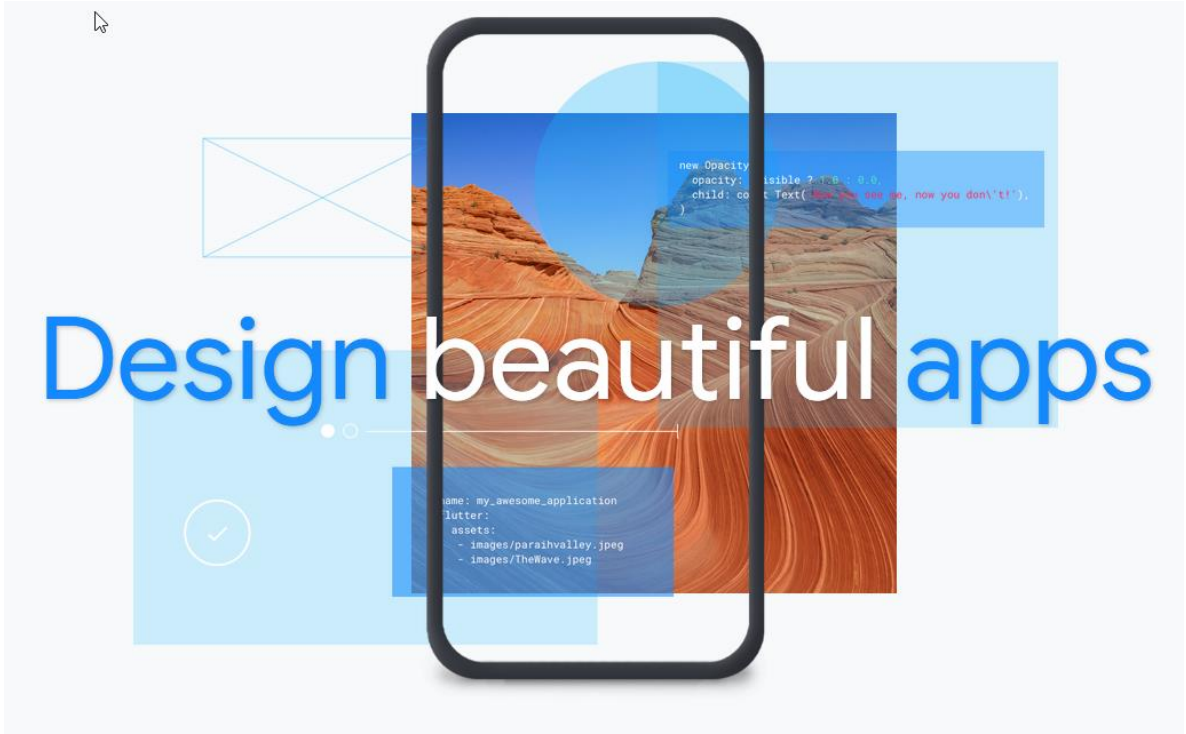
Transformando o futuro das pessoas
e as pessoas para o futuro.

#Senacfaz75

Senac

Desenvolvimento Mobile: Flutter



O que é o Flutter?

Flutter é um kit de desenvolvimento de interface de usuário (UI toolkit), de código aberto, criado pelo Google, que possibilita a criação de aplicativos compilados nativamente. Atualmente pode compilar para Android, iOS, Windows, Mac, Linux, Google *Fuchsia e Web.

O que é o Dart?

Os aplicativos Flutter são escritos na linguagem de programação Dart e fazem uso de muitos dos recursos mais avançados da linguagem.

No Windows, macOS e Linux, por meio do projeto semi-oficial Flutter Desktop Embedding, o Flutter é executado na máquina virtual Dart, que possui um mecanismo de compilação que ocorre em tempo de execução. Ao escrever e depurar um aplicativo, o Flutter usa a compilação JIT, permitindo o "hot reload", com a qual as modificações nos arquivos de origem podem ser injetadas em um aplicativo em execução. O Flutter estende isso com suporte para hot reload de widgets stateful, onde na maioria dos casos as alterações no código-fonte podem ser refletidas imediatamente no aplicativo em execução, sem a necessidade de uma reinicialização ou perda do Estado.

As versões de lançamento dos aplicativos Flutter são compiladas com a compilação antecipada (AOT) no Android e no iOS, possibilitando o alto desempenho do Flutter em dispositivos móveis.

*Fuchsia é um sistema operacional atualmente sendo desenvolvido pelo Google. Ao contrário de sistemas operacionais anteriores desenvolvidos pelo Google, como o Chrome OS e o Android, que são baseados no kernel Linux, Fuchsia é baseado em um novo microkernel chamado Zircon (o nome anterior era Magenta), derivado do Little Kernel, que foi destinado para sistemas embarcados e é principalmente escrito em C. Fuchsia foi projetado para ser executado em uma infinidade de dispositivos, incluindo telefones celulares e computadores pessoais.

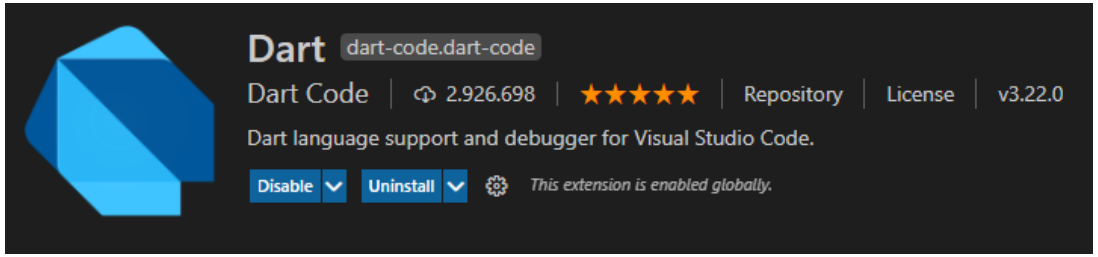
Instalações:

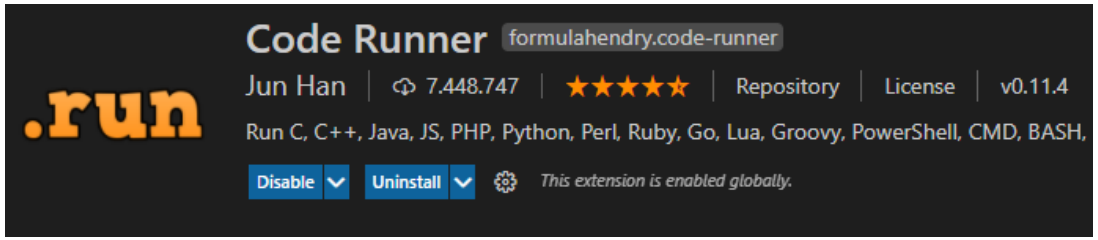
Java SE
https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html

Flutter
https://flutter.dev/docs/get-started/install/windows

OBS: Colocar a pasta bin do Flutter no Path do Windows

Extensões VS Code





Null Safety

A partir do Dart 2.12, temos a possibilidade de usar o Null Safety para facilitar nossas aplicações, dificultando erros de valores nulos. Aprenderemos a identificar erros de valores nulos, aplicar no nosso projeto do ByteBank, atualizar o Dart para a nova versão e migrar um projeto inteiro! Assim podemos criar códigos complexos com mais facilidade!

Erros de valor Null

Existem infinitos erros que podem acontecer por conta de um valor nulo:

- Você pode estar esperando um dado do seu back-end, mas ele não existe ainda...
- Você pode criar uma lista que muda de tamanho de acordo com a quantidade de produtos...

Fonte: Kako(Caio couto Moreira). Alura. Disponível em: https://www.alura.com.br/artigos/flutter-null-safety

Aula 1 – Hello dart e tipos de variáveis!

hellodart.dart > ...

```dart
//É preciso criar uma classe main para rodar o código Dart
Run | Debug
main() {
  //print é o comando de saída em Dart
  print('Hello dart!');
  print('Como vai?');

  //Tipos de variáveis em Dart
  //Dart faz inferência de tipo (Genérico)
  var nome = 'John Doe';
  //nome é do tipo String
  //Se eu tentar colocar outro valo nessa variável que não seja String
  //vai dar erro
  //var nome = 10; //Tipo já foi definido como String (ERRO)

  /**
   * Podemos também definir uma variável
   * já com o seu tipo
   */

  //Variável do tipo String
  String novoNome = 'Jane Doe'; //Estou tipando minha variável
  String CPF = '999.999.999.99-99';

  //Definindo um tipo numérico inteiro
  int ano = 1970;

  //Definindo um número decimal
  double altura = 1.77;

  //Definindo um tipo booleano (Verdadeiro ou falso)
  bool vf = true;
  bool fv = false;

  //Saída Interpolada, ao contrário do javascript
  //Não é reciso as {} e `` para interpolar valores
  print('Seu nome: $novoNome'); //Lembra PHP
  print('Nascimento: $ano');
  print('Sua altura: $altura');
  print('Resultado VF: $vf');
  print('Resultado FV: $fv');
}
```

```
Hello dart!
Como vai?
Seu nome: Jane Doe
Nascimento: 1970
Sua altura: 1.77
Resultado VF: true
Resultado FV: false

[Done] exited with code=0 in 1.359 seconds
```

Aula 1 – Estruturas de Dados Arrays (arrays.dart)

```dart
Run | Debug
void main() {
  //Arays são estruturas que armazenam mais de um valor
  //a uma variável

  //Criando um array
  var compras = ['Macarrão', 'Feijão', 'Pão', 'Manteiga'];

  //Exibindo o array
  print('Lista de compras: $compras');

  //Exibindo os itens pelo índice
  print('Nome do primeiro produto: ${compras[0]}');
  print('Nome do segundo produto: ${compras[1]}');
  print('Nome do terceiro produto: ${compras[3]}');

  //Item out of range
  //print('Nome do primeiro produto: ${compras[5]}');

  //Acessando e alterando um valor do array
  compras[0] = 'Arroz';
  //Exibindo os itens pelo índice
  print('Nome do primeiro produto: ${compras[0]}');

  //Criando array numérico
  var pares = [0, 2, 4, 6];

  //Exibindo o array
  print('Lista de números pares $pares');
  print('------------------------------------------------------');

  //Métodos utilizados em arrays
  /**
   * first() : Retorna o primeiro elemento do Array
   * last() : Retorna o último elemento do Array
   * isEmpty() : Retorna true se a lista está vazia, caso contrário, false.
   * length() : Retorna o tamanho do Array
   */

  var listaNomes = ['José Maria', 'Pedro da Silva', 'Cristina Pereira'];

  print('Primeiro nome: ${listaNomes.first}');
  print('Último nome: ${listaNomes.last}');
  print(
      'O Array está vazio? ${listaNomes.isEmpty}');
  print('Tamanho do Array: ${listaNomes.length}');
}
```

```
Lista de compras: [Macarrão, Feijão, Pão, Manteiga]
Nome do primeiro produto: Macarrão
Nome do segundo produto: Feijão
Nome do terceiro produto: Manteiga
Nome do primeiro produto: Arroz
Lista de números pares [0, 2, 4, 6]
------------------------------------------------------
Primeiro nome: José Maria
Último nome: Cristina Pereira
O Array está vazio? false
Tamanho do Array: 3
Exited
```

Aula 1 – Operdaores Aritméticos (oparitimeticos.dart)

```dart
Run | Debug
void main() {
  //Operadores aritméticos
  // + - * / %

  int a = 20;
  int b = 5;

  //Operador de Soma
  var soma = a + b;

  //Operador de Subtração
  var subt = a - b;

  //Operador de Multiplicação/Produto
  var produto = a * b;

  //Operador de divisão
  var divisao = a / b;

  //operador resto da divisão
  var restoDiv = a % b;

  //Saída
  print('Soma de $a + $b = $soma');
  print('Subtração de $a - $b = $subt');
  print('Produto de $a * $b = $produto');
  print('Divisão de $a / $b = $divisao');
  print('Resto da Divisão de $a % $b = $restoDiv');

  //Operdador resumido
  int num = 100;
  num += 20;
  num -= 10; //Pode ser * /

  //Saída
  print('Número: $num');

  //Incremento e decremento
  num++;
  num--;

  //Saída
  print('Número: $num');
}
```

```
Soma de 20 + 5 = 25
Subtração de 20 - 5 = 15
Produto de 20 * 5 = 100
Divisão de 20 / 5 = 4.0
Resto da Divisão de 20 % 5 = 0
Número: 110
Número: 110

[Done] exited with code=0 in 1.345 seconds
```

Aula 1 – Operadores Relacionais (oprelaconais.dart)

```dart
Run | Debug
void main() {
  //Operdaores Relacionais
  /**
   * ==, != (Igual e diferente)
   * >, < (Maior e menor)
   * >=, <= (Maio igual e Menor igual)
   */

  //Declarando variáveis
  int a = 20;
  int b = 5;

  //Verificando as variáveis
  print('$a = $b? Resultado: ${a == b}');
  print('$a ≠ $b? Resultado: ${a != b}');
  print('$a > $b? Resultado: ${a > b}');
  print('$a < $b? Resultado: ${a < b}');
  print('$a ≥ $b? Resultado: ${a >= b}');
  print('$a ≤ $b? Resultado: ${a <= b}');

  //Podemos atribuir esses resultados a uma variável
  bool igual = a == b;
  print('Verificação de igualdade: $igual');
}

```

```
20 = 5? Resultado: false
20 ≠ 5? Resultado: true
20 > 5? Resultado: true
20 < 5? Resultado: false
20 ≥ 5? Resultado: true
20 ≤ 5? Resultado: false
Verificação de igualdade: false

[Done] exited with code=0 in 1.508 seconds
```

Aula 1 – Operadores Lógicos (oplogicos.dart)

```dart
Run | Debug
void main() {
  //Operadores Lógicos
  /**
   * && (E) --> V && V = V
   * || (OU) --> F || F = F
   * ! Negação
   */

  //Declarando as variáveis
  int a = 20;
  int b = 5;
  int c = 7;

  //Verificando Verdadeiro
  bool proposicao1 = a > b;
  bool proposicao2 = b < c;

  //Saída
  //        V         V
  print('$a > $b && $b < $c - Resposta: ${proposicao1 && proposicao2}');
  //        V         F
  print('$a > $b || $b > $c - Resposta: ${proposicao1 || proposicao2}');

  //Verificando o Falso
  bool proposicao3 = a < b;
  bool proposicao4 = b > c;

  //Saída
  print('$a < $b && $b > $c - Resposta: ${proposicao3 && proposicao3}');
  print('$a > $b || $b > $c - Resposta: ${proposicao4 || proposicao4}');

  //Negar um valor
  bool v = true;
  bool f = false;

  //Saída
  print('Negando o V: ${!v}');
  print('Negando o F: ${!f}');
}
```

```
20 > 5 && 5 < 7 - Resposta: true
20 > 5 || 5 > 7 - Resposta: true
20 < 5 && 5 > 7 - Resposta: false
20 > 5 || 5 > 7 - Resposta: false
Negando o V: false
Negando o F: true

[Done] exited with code=0 in 1.404 seconds
```

Aula 1 – Condicional If Else (condicionais.dart)

```dart
Run | Debug
void main() {
  //Condicionais
  //Declaração de variáveis
  int a = 10;
  int b = 5;

  //Condicional simples, Else opcional
  if (a > b) {
    print('Informação Verdadeira!');
  } else {
    print('Informação Falsa');
  }

  //Declarando Variável
  String nome = 'john';

  //Condicional
  if (nome != 'Jane') {
    print('Os nomes são diferentes!');
  } else {
    print('Os nomes são iguais!');
  }

  //Condicionais encadeadas
  //Testando mais de uma condição
  //Declarando variável
  double media = 4;

  //Condicional
  if (media >= 7.5 && media <= 10) {
    print('O aluno passou de ano!');
  } else if (media >= 5 && media < 7.5) {
    print('Aluno em recuperação!');
  } else {
    print('Aluno Reprovado!');
  }

}
```

```
Informação Verdadeira!
Os nomes são diferentes!
Aluno Reprovado!
Exited
```

Aula 1 – Condicional Switch (condswitch.dart)

```dart
Run | Debug
void main() {
  print('1 - Numero par');
  print('2 - Maior número');
  print('3 - Sair');

  int opcao = 2;

  switch (opcao) {
    case 1:
      //Declaração
      int numero = 10;

      //Condicional
      if (numero % 2 == 0) {
        print('O número $numero é par!');
      } else {
        print('O número $numero é ímpar!');
      }
      break;

    case 2:
      //Declaração
      int a = 3;
      int b = 3;

      //Condicional
      if (a > b) {
        print('O número $a é maior que o número $b!');
      } else if (a < b) {
        print('O número $a é menor que o número $b!');
      } else {
        print('Os números são iguais!');
      }
      break;

    default:
      print('Fora do intervalo de opções!');
  }
}
```

```
1 - Numero par
2 - Maior número
3 - Sair
Os números são iguais!
Exited
```

Aula 1 – Loop While (while.dart)

```dart
//Loop While
Run | Debug
/**
 * O loop while executa as instruções cada vez que a
 * condição especificada é avaliada como verdadeira. Em outras palavras,
 * o loop avalia a condição antes que o bloco de código seja executado
 */
void main() {
  //Declarando uma Flag
  int contador = 0;

  while (contador <= 20) {
    print('Número: $contador');

    //Incrementando o contaador para o loop não ficar infinito
    contador++;

    //condicional para quebra o Loop
    //Podemos usar também o continue
    if (contador == 10) {
      print('Loop interrompido!');
      break;
    }
  }
}
```

```
Número: 0
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Loop interrompido!
Exited
```

Aula 1 – Loop For (for.dart)

```dart
//Loop For
Run | Debug
/**
 * O laço For é uma implementação de um loop definido.
 * O loop for executa o bloco de código por um determinado número de vezes.
 * Ele pode ser usado para iterar sobre um conjunto fixo de valores,
 * como uma matriz.
 */

void main() {
  //Declaração
  int contador = 20;

  for (var i = 0; i < contador; i++) {
    print('Número: $i');
    if (i == 10) {
      print('Loop interrompido!');
      break;
      //Podemos usar o continue também
    }
  }
  print('------------------------------------------------');

  //Varrendo array
  //Definindo o Array
  var nomes = ['Bete', 'Ana', 'Pedro', 'João', 'Maria'];

  for (var i = 0; i < nomes.length; i++) {
    print('Nome: ${nomes[i]}');
  }
  print('------------------------------------------------');

  //Simulando Post de Filmes
  var filmes = [
    'Matrix',
    'Uma vida iluminada',
    'Teoria de Tudo',
    'Divertidamente'
  ];

  for (var filme in filmes) {
    print(filme);
  }
}
```

```
Número: 0
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Número: 10
Loop interrompido!
------------------------------------------------
Nome: Bete
Nome: Ana
Nome: Pedro
Nome: João
Nome: Maria
------------------------------------------------
Matrix
Uma vida iluminada
Teoria de Tudo
Divertidamente

[Done] exited with code=0 in 2.107 seconds
```

Aula 2 – Funções (funcao_void.dart)

```dart
//Main() é uma função de inicialização de código dart
//Essa função é obrigatória
//Essa função é void, não retorna valor

//Exemplos
void mostrarNome() {
  print('John Doe');
}

void linha() {
  print('--------------------------------');
}

//Função com parâmetro
void calcularValores(String operador, double a, double b) {
  if (operador == '+') {
    double soma = a + b;
    print('Operador \"$operador\" a soma de $a + $b = $soma');
  } else if (operador == '*') {
    double produto = a * b;
    print('Operador \"$operador\" o produto de $a x $b = $produto');
  } else if (operador == '-') {
    double subt = a - b;
    print('Operador \"$operador\" a subtração de $a - $b = $subt');
  } else if (operador == '/') {
    double divisao = a / b;
    if (b == 0) {
      print('Divisão inválida!');
    } else {
      print('Operador \"$operador\" o produto de $a / $b = $divisao');
    }
  }
}

```

```dart
void main() {
  //Chamando a função
  mostrarNome();
  linha();

  calcularValores('+', 10, 5);
  linha();

  calcularValores('-', 3, 5);
  linha();

  calcularValores('*', 30, 2);
  linha();

  calcularValores('/', 5, 10);
  linha();
}

```

```
John Doe
--------------------------------
Operador "+" a soma de 10.0 + 5.0 = 15.0
--------------------------------
Operador "-" a subtração de 3.0 - 5.0 = -2.0
--------------------------------
Operador "*" o produto de 30.0 x 2.0 = 60.0
--------------------------------
Operador "/" o produto de 5.0 / 10.0 = 0.5
--------------------------------

[Done] exited with code=0 in 1.336 seconds
```

Aula 2 – Funções (funcao_retorno.dart)

```dart
//Funções com retorno

//Declaração
//Foi importado assim que eu chamei a função linha()
import 'funcao_void.dart';

double calcularMedia(double n1, double n2, double n3, double n4) {
  double media = (n1 + n2 + n3 + n4) / 4;
  return media;
}

double calcularDivisao(double n1, double n2) {
  double div = (n1 / n2);
  return div;
}

 //Função Resumida
  double calcularPorcentagem(percent, valor) => (percent * valor) / 100;

Run | Debug
void main() {
  double media = calcularMedia(10, 10, 10, 10);
  double divisao = calcularDivisao(10, 5);

  //Saída
  print('Média Aritmética');
  print('A média é $media');
  linha();
  print('A divisão é $divisao');
  linha();

////////////////////////////////////////////////////////////////////////

  //Declaração de variáveis
  double percent = 5;
  double valor = 100;

  //Executa a função e guarada o valor em uma variável
  double percentual = calcularPorcentagem(percent, valor);

  //Imprime 10% de 100
  print('$percent% de $valor = $percentual');
}
```

```
Média Aritmética
A média é 10.0
--------------------------------
A divisão é 2.0
--------------------------------
5.0% de 100.0 = 5.0
Exited
```

Aula 2 – Funções (funcao_param_opcional.dart)

```dart
//Parâmetro opcional

import 'dart:math';

Run | Debug
void main() {
  //declarar limite
  int numero = sortearNumero(1000);
  //Saída
  print('O número sorteado foi: $numero');

  ////////////////////////////////////////////////////
  int valor1 = 10;
  int valor2 = 20;

  //Segundo parâmetro é opcional
  int soma = somarValores(valor1, valor2);

  //Saída
  print('Resultado dos valores somados: $soma');
}

//Utilizamos os colchetes para determinar o parâmetro opcional
int sortearNumero([int limite = 3]) {
  //Se nenhum valor limite for passado
  //Será assumido o 3
  return Random().nextInt(limite);
}

//1 parâmetro obrigatório e 1 opcional
int somarValores(int v1, [int v2 = 0]) {
  //Se não informar o valor 2
  //o parâmetro assume 0
  print('Valor 1: $v1');
  print('Valor 2: $v2');
  return v1 + v2;
}
```

```
O número sorteado foi: 778
Valor 1: 10
Valor 2: 20
Resultado dos valores somados: 30
Exited
```

Aula 2 – Funções (funcao_param_nomeado_1.dart)

```dart
//Parâmetros nomeados em função
//ajuda na clareza do código
//Evitando confusão com os parâmetros posicionais

Run | Debug
void main() {
  //chamando a função com parâmetros nomeados
  exibirCadastro(salario: 1500, funcionario: 'John Doe', funcao: 'Gerente');
}

//Funções
exibirCadastro(
  //Utilizamos required por conta da nova funcionalidade do Dart
  //Assunto complexo: non-null
    {required String funcao,
    required double salario,
    required String funcionario}) {

  print('Nome do Funcionário: $funcionario');
  print('Função: $funcao');
  print('Salário: R\$$salario');
  return 'Tudo ok!';
}
```

```
Nome do Funcionário: John Doe
Função: Gerente
Salário: R$1500.0
Exited
```

Aula 2 – Funções (funcao_param_nomeada_2.dart)

```dart
//Função anônima e parâmetros opcionais
//Função com parâmetros opcionais
void avaliarFilme(String nomeFilme, {categoria, nota}) {

  //Verificar se o valor está nulo
  var n = nota ?? 0;
  var cat = categoria ?? 'Sem categoria';

  //Imprime os Dados
  print('Nome do Filme: $nomeFilme');
  print('Categoria: $cat');
  print('Nota: $n');

}

Run | Debug
void main() { // função obrigatória

  //Executar a função avaliarFilme()
  //avaliarFilme('Matrix');
  avaliarFilme('Matrix', categoria: 'Ficção', nota: 10);

}
```

```
Nome do Filme: Matrix
Categoria: Ficção
Nota: 10
Exited
```

Aula 2 – Funções (funcao_anonima.dart)

```dart
//Funções anônimas: Funções que podem ser
//armazenadas dentro de variáveis

import 'dart:math';

Run | Debug
void main() {
  //Criando e definindo o tipo

  //Minha função é do tipo int
  int Function(int, int) soma = somarFunction;

  //Minha função é do tipo int/double/String/Dynamic
  String Function(String, String) mostraTexto = mostrarTexto;

  //Uma outra forma de criar a função anônima
  //Minha função é do tipo int/double/String/Dynamic
  dynamic Function(double, double) potencia = (base, expoente) {
    return pow(base, expoente);
  };//Não esquecer

  //mais uma forma de criar esse tipo de função
  var produto = (double a, int b) {
    return a * b;
  }; //Não esquecer

  //Saída
  print('A soma dos valores é ${soma(10, 10)}');
  print('Frase: ${mostraTexto("Olá", " Mundo!")}');
  print('Resultado da potência é:  ${potencia(2, 3)}');
  print('Resultado da multiplicação é:  ${produto(10, 3)}');
}

//Funções
int somarFunction(int a, int b) {
  return a + b;
}

String mostrarTexto(String a, String b) {
  return a + b;
}
```

```
A soma dos valores é 20
Frase: Olá Mundo!
Resultado da potência é:  8.0
Resultado da multiplicação é:  30.0
Exited
```

Aula 2 – Funções (funcao_arrow.dart)

```dart
//Arrow function é uma forma reduzida de representar
//um função. Contra: são limitadas, não podem ser utilidas
//em todas as situações

Run | Debug
void main() {
  //Função anônima
  var produto = (int a, int b) {
    return a * b;
  };

  //Arrow Function
  //1 sentença de código apenas
  var divisao = (double a, double b) => a / b;
  var modulo = (double c, double d) => c % d;
  var media = (double n1, double n2, double n3) => (n1 + n2 + n3) / 3;

  //Saída
  print('O valor do produto é: ${produto(10, 5)}');
  print('O valor da divisão é: ${divisao(50, 2)}');
  print('O valor da divisão é: ${modulo(5, 3)}');
  print('A média calculada é: ${media(4, 5, 6)}');
}
```

```
O valor do produto é: 50
O valor da divisão é: 25.0
O valor da divisão é: 2.0
A média calculada é: 5.0
Exited
```

Aula 2 – Funções (funcao_tipo_dinâmico.dart)

```dart
//Dart é altamente tipado
//Cuidado ao criar suas variáveis
Run | Debug
void main() {

  //Criando tipos dinâmicos
  dynamic a = 1000;
  dynamic b = ' Mundo!';

  //Chamada da função e atribuição em uma variável
  dynamic uniao = concatenar(a, b);

  //Saída
  print('A união dos valores \"$a\" e \"$b\" é: $uniao');
}

//Função para juntar os valores de a e b
dynamic concatenar(param1, param2) {

  //método toString faz casting
  print(param1.toString() + param2.toString());

  //Retornando os valores
  return param1.toString() + param2.toString();
}
```

```
1000 Mundo!
A união dos valores "1000" e " Mundo!" é: 1000 Mundo!
Exited
```

Aula 2 – Funções (funcao_param1.dart)

```dart
//As funções pode também receber como parâmetros outroas funções

//Importando a biblioteca matemática
import 'dart:math';

Run | Debug
void main() {
  //Calcular par ou ímpar
  var par = () => print('Número Par');
  var impar = () => print('Número Ímpar');

  //Chamndo a função calcularParImpar
  calcularParImpar(par, impar);
}

//Funções
void calcularParImpar(Function calcularPar, Function calcularImpar) {
  //Criando número randômico
  //O VsCode importa da Biclioteca dar:math automáticamente
  var numero = Random().nextInt(50);
  print('Número randômico: $numero');
  //Verificando
  if (numero % 2 == 0) {
    calcularPar();
  } else {
    calcularImpar();
  }
}
```

```
Número randômico: 1
Número Ímpar
Exited
```

Aula 2 – Funções (funcao_param2.dart)

```dart
//Importando a biblioteca matemática
import 'dart:math';

//Criando uma função para encontrar o Delta
double calcularDelta(double a, double b, double c) {
  //Fórmula do Delta = b² - 4ac
  double delta = (b * b) - (4 * a * c);
  return delta;
}

//Criando uma função para encontrar os valores de X1 e X2
//Passo os valores das incógnitas e a função do Delta
dynamic calcularEquacao(double a, double b, double c, Function calcularDelta) {
  //Lista para a solucao
  var solucao = [];

  //Calculando as raízes
  var x1 = ((-b) + sqrt(calcularDelta(a, b, c))) / (2 * a);
  var x2 = ((-b) - sqrt(calcularDelta(a, b, c))) / (2 * a);

  //Inserindo os valores na lista
  solucao.add(x1);
  solucao.add(x2);
  return solucao;
}

Run | Debug
void main() {
  //Declarando as incógnitas
  double a = 1;
  double b = 2;
  double c = -15;

  //Chamando a função com parâmetros misturados
  var equacao = calcularEquacao(a, b, c, calcularDelta);

  //Saída
  print(equacao);
}
```

```
[3.0, -5.0]
Exited
```

Aula 2 – Funções (funcao_ret_funcao.dart)

```dart
//Com uma função retornando outra função
//podemos fazer com que alguma parte do nosso código
//seja executada em um outro momento
double Function(double) calcularJuros(double a) {

  //Função interna
  return (double b) { //Retornando uma função anônima
    return a * b;
  };

}

Run | Debug
void main() {

  //Passando os dois valores, a e b
  print(calcularJuros(2)(20));

  //Colocando a função em uma variável para otimizar o processo
  //passando 1 valor
  var juros = calcularJuros(.10);

  //Saída passando o valor de b
  print(juros(1500));
  print(juros(1000));
  print(juros(10000));
}
```

```
40.0
150.0
100.0
1000.0
Exited
```

Aula 2 – Funções (lista_2.dart)

```dart
void main() {
  //Criando uma lista
  var minhaLista = [];

  //Método para inserir valor na lista
  minhaLista.add('Dart');
  minhaLista.add('Java');
  minhaLista.add('Python');
  minhaLista.add('Ruby');
  minhaLista.add('Php');
  minhaLista.add('JavaScript');

  //Método para retornar o tamanho da lista
  int tamanhoLista = minhaLista.length;

  //Método para remover elementos da lista
  minhaLista.remove('Php');

  //Método para remover elementos da lista pelo índice
  minhaLista.removeAt(2);

  //Método para remover o último elemento da lista
  minhaLista.removeLast();

  //Método para remover os elementos pelo intervalo
  minhaLista.removeRange(1, minhaLista.length);

  //Imprimindo a Lista
  print('Minha lista: $minhaLista');
  print('Tamanho da lista: $tamanhoLista');
}
```

```
Minha lista: [Dart]
Tamanho da lista: 6
Exited
```

Aula 2 – Funções (filtragem_1.dart)

```dart
//Exemplo de filtragem sem o uso do método dart
//Verrendo os elemento de uma lista e colocando em outra
Run | Debug
void main() {

  //Criando a lista de valores
  var valores = [5.5, 7.0, 6.3, 9.7, 10.0, 3.5];

  //Criando uma lista para guardar as valores maiores que 7
  var valoresAcimaMedia = [];

  //for para varrer a lista e pegar as valores
  for (var valor in valores) {
    if (valor >= 7) {

      //Método add() para guardar as valores em uma lista nova
      valoresAcimaMedia.add(valor);
    }
  }

  //Imprimindo da lista
  print(valores);
  print(valoresAcimaMedia);
}
```

```
[5.5, 7.0, 6.3, 9.7, 10.0, 3.5]
[7.0, 9.7, 10.0]
Exited
```

Aula 2 – Funções (filtragem_2.dart)

```dart
//Melhorando a filtragem de dados
//Agora vamos usar o método Dart para fazer a filtragem
Run | Debug
void main() {
  //Criando a lista de valores
  var valores = [5.5, 7.0, 6.3, 9.7, 10.0, 3.5];

  //Criar uma função para pegar as valores
  //Veja que o retorno de arrow function é um boolean
  //É desta forma que o método where funciona
  bool Function(double) valoresAcimaMedia = (double valor) => valor >= 6;

  //Usando o Método Where
  //De todas as valores pegue somente as acima da média
  var valoresNovos = valores.where(valoresAcimaMedia);
  print('Primeira lista: $valores');
  print('Lista nova: $valoresNovos');
}
```

```
Primeiro lista: [5.5, 7.0, 6.3, 9.7, 10.0, 3.5]
Lista nova: (7.0, 6.3, 9.7, 10.0)
Exited
```

Aula 2 – Funções (filtragem_3.dart)

```dart
//Simulando o a implementação Where
//Para filtrarValores valores de forma genérica

//Criar uma função
//funcao pode ter qualquer nome
List<double> filtrarValores(List<double> lista, bool Function(double) funcao) {
  List<double> listaFiltrada = [];

  //Usar um laço para inserir os elementos na lista
  //de acordo com a condição passada para função filtrarValores
  for (double elemento in lista) {
    if (funcao(elemento)) {

      //Inserindo o valor na lista
      listaFiltrada.add(elemento);
    }
  }
  return listaFiltrada;
}

Run | Debug
void main() {
  //Criar um array Double
  var valores = [10.5, 9.4, 15.7, 6.3, 7.0];

  //Criar uma função para montar a condição de varredura
  //Só preenche com valores maiores iguais a 8.0
  var valoresAcimaMedia = (double valor) => valor >= 8.0;

  //Chama a função que vai filtrarValores nossos valores
  var executaFiltragem = filtrarValores(valores, valoresAcimaMedia);

  //Saída
  print('Lista original: $valores');
  print('Lista nova: $executaFiltragem');
}
```

```
Lista original: [10.5, 9.4, 15.7, 6.3, 7.0]
Lista nova: [10.5, 9.4, 15.7]
Exited
```

Aula 2 – Funções (funcao_map.dart)

```dart
//Map: uma lista mapeada sempre terá o mesmo número de elementos
//Ao contrário do where, onde podemos obter listas de diferentes tamanhos
//Map serve para mapear um elemento em outro elemento
//Exemplo: em uma lista de nomes podemos obter outra lista com
//o tamanho (em caracteres) de cada nome.
import 'funcao_void.dart';
Run | Debug
void main() {
  //Criando um Map
  var mediaAlunos = [
    {'nome': 'John', 'nota': 8},
    {'nome': 'Jane', 'nota': 7},
    {'nome': 'Carol', 'nota': 8},
    {'nome': 'Mike', 'nota': 6},
  ];

//Uma função que pega os alunos pelo nome
//Map aqui é somente um tipo, assim como List
  String Function(Map) capturaAluno = (aluno) => aluno['nome'];

//Método map() para mapear o resultado da função
//Retornarno para um variável
  var listaResultado = mediaAlunos.map(capturaAluno);

  //Veja que o resultado tem o mesmo tamanho de nosso Map
  print('Lista de Nomes: $listaResultado');
  print('O Map tem tamanho ${mediaAlunos.length}');

  //Outro exemplo no uso de Map
  //Para mapear uma lista
  List<int> numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  //Recuperar um lista com o dobro de cada valor
  var resultadoDobro = numeros.map((numero) => numero * 2);

  linha();
  print(resultadoDobro);
}
```

```
Lista de Nomes: (John, Jane, Carol, Mike)
O Map tem tamanho 4
------------------------------------------
(2, 4, 6, 8, 10, 12, 14, 16, 18, 20)
Exited
```

Aula 2 – Funções (funcao_reduce.dart)

```dart
import 'funcao_void.dart';

Run | Debug
void main() {
  //Uma forma noraml para calcular a média
  var notasAlunos = [10, 6, 8, 7];

  //Preciso colocar 0.0 para a inferência funcionar
  //Caso contrário, devo explicitar o tipo (double)
  var totalNotas = 0.0;

  //percorrendo a lista com for
  //Pacada cada nota em notas de alunos, some as notas
  for (var nota in notasAlunos) {
    totalNotas += nota;
  }

  //Média
  double media = totalNotas / 4;

  //Saída
  linha();
  print('MÉTODO CONVENCINAL');
  print('===========================================');
  print('Valor total das notas: $totalNotas');
  print('Média de notas: $media');
  linha();
  print('MÉTODO REDUCE()');

```

```dart
  ////////////////////////////////////////////////////////////
  //MÉTODO UTILIZANDO O REDUCE
  /**
   * Reduz uma coleção a um único valor combinando iterativamente os
   * elementos da coleção usando a função fornecida.
   * O iterável deve ter pelo menos um elemento. Se tiver apenas um elemento,
   * esse elemento é retornado.
   * Caso contrário, este método começa com o primeiro elemento do iterador e,
   * em seguida, combina-o com os elementos restantes na ordem de iteração.
   */

  //Declarando uma lista de valores double
  var precoPlacaVideo = [15000.99, 23000.99, 16000.99, 19000.99];

  //O reduce() é mais interessante que o Map, pois podemos trasformar
  //no lista no que quisermos. O reduce() pede uma função para determinar
  //o que vais fazer
  var precoTotal = precoPlacaVideo.reduce(funcaoSomarValores);

  //Imprimindo o resultado

  print('===========================================');
  print('Soma dos valores: $precoTotal');

  ///OUTRO EXEMPLO

  //Declarando minha lista
  var listaAlunos = ['Maria', 'José', 'Pedro'];

  //usar reduce() para juntar os nomes
  String alunosCombinados =
      listaAlunos.reduce((valor, elemento) => "$valor | $elemento");

  print('Lista combinada: $alunosCombinados');
}

```

Aula 2 – Funções (funcao_reduce.dart)

```dart

//Criar função para o reduce
//Veja que ele pede uma função:
//double reduce(double Function(double, double) combine)
double funcaoSomarValores(double a, double b) {
  //O que acontece
  print('Valor de $a + $b');

  //Retorno
  return a + b;
}
```

```
-----------------------------------------------
MÉTODO CONVENCINAL
===============================================
Valor total das notas: 31.0
Média de notas: 7.75
-----------------------------------------------
MÉTODO REDUCE()
Valor de 15000.99 + 23000.99
Valor de 38001.98 + 16000.99
Valor de 54002.97 + 19000.99
===============================================
Soma dos valores: 73003.96
Lista combinada: Maria | José | Pedro
Exited
```

Aula 2 – Funções (funcao_reduce.dart)

```dart
//Exemplo na utilização do map(), reduce(), where()

import 'funcao_void.dart';

Run | Debug
void main() {
  //Declarar uma lista de nomes e valores
  var listaNomeValor = [
    {'nome': 'John', 'valor': 100.5},
    {'nome': 'Jane', 'valor': 200.5},
    {'nome': 'Carol', 'valor': 300.5},
    {'nome': 'Mike', 'valor': 400.5},
  ];

  //Pega Valores
  var ValorFinal = listaNomeValor
      .map((nome) => nome['valor']) //Cria uma lista de valores
      //Faz um casting no valor
      .map((valor) => (valor as double).roundToDouble())
      .where((valor) => valor >= 200.0); //Cria uma lista com critério

  //Soma os valores da lista filtrada pelo Where
  var total = ValorFinal.reduce((valor, elemento) => valor + elemento);

  //Calcula a média
  var media = total / ValorFinal.length;

  //Saída
  linha();
  print('A média de valores é: $media');
  linha();
}
```

```
--------------------------------------------
A média de valores é: 301.0
--------------------------------------------
Exited
```

Programação Orientada a Objetos

Aula 3 – Orientação a objetos

Programação Orientada a Objetos (também conhecida pela sua sigla POO) é um modelo de análise, projeto e programação de software baseado na composição e interação entre diversas unidades chamadas de 'objetos'. A POO é um dos 4 principais paradigmas de programação (as outras são programação imperativa, funcional e lógica). Os objetos são operados com o conceito de 'this' (isto) ou 'self' (si), de forma que seus métodos (muitas vezes) modifiquem os dados da própria instância.

Os programas são arquitetados através de objetos que interagem entre si. Dentre as várias abordagens da POO, as baseadas em classes são as mais comuns: objetos são instâncias de classes, o que em geral também define o tipo do objeto.

Cada classe determina o comportamento (definido nos métodos) e estados possíveis (atributos) de seus objetos, assim como o relacionamento com outros objetos.

A alternativa mais usual ao uso de classes é o uso de protótipos. Neste caso, objetos são cópias de outros objetos, não instâncias de classes. Javascript e Lua são exemplos de linguagens cuja POO é realizada por protótipos. A diferença prática mais evidente é que na POO baseada em protótipos apenas a herança simples é implementada pela cópia do objeto. Assim, na POO, implementa-se um conjunto de classes passíveis de serem instanciadas como objetos, e.g. Python e C++ (ou objetos protótipos que são copiados e alterados, e.g. JavaScript e VimL).

Críticas à POO

As críticas mais recorrentes à POO podem ser resumidas em:

1) não satisfazer os objetivos de reusabilidade e modularidade, e
2) a ênfase demasiada em design e modelamento de software em detrimento de outros aspectos (computabilidade/algoritmos).
3) Não há consenso de que a POO realmente seja útil para modelar a realidade, tampouco se modelar a realidade é um objetivo pertinente. A premissa de que 'a POO é apropriada para modelar sistemas complexos com comportamentos complexos', contrasta com o renomado princípio KISS (mantenha simples). Linguagens naturais não compartilham da estratégia usual na POO: 'priorizar coisas ao invés de ações', o que leva muitas vezes a representações contorcidas.
4) A POO muitas vezes é realizada sem uma representação transparente do fluxo de controle (ordem das instruções), o que tem se tornado uma desvantagem com a relevância, acentuada nos últimos anos, do processamento paralelo. Outras críticas usuais incluem: a POO é 'intrinsecamente menos eficiente' que o código procedural, em geral demora mais para compilar, e tende a ser extremamente complexa

Críticas de programadores notórios

Joe Armstrong: "O problema com linguagens orientadas a objetos é que você tem todo esse ambiente implícito que elas levam consigo. Você queria uma banana mas o que você conseguiu foi um gorila segurando a banana e uma selva inteira."

Alexander Stepanov: "Dizer que tudo é um objeto é simplesmente dizer nada."

Steve Yegge: "Por que alguém iria tão longe para colocar um só aspecto da fala [substantivos] em um pedestal?"

Eric Steven Raymond: "A POO encoraja programas com muitas camadas e destrói a transparência."

Rob Pike: "A POO constitui os números romanos da computação."

Aula 3 – Linguagens de Programação

Procedural:
O termo Programação procedural (ou programação procedimental) é às vezes utilizado como sinônimo de Programação imperativa (Paradigma de programação que especifica os passos que um programa deve seguir para alcançar um estado desejado), mas pode se referir (como neste artigo) a um paradigma de programação baseado no conceito de chamadas a procedimento (en: procedure call). Os Procedimentos, também conhecidos como rotinas, subrotinas, métodos, ou funções (que não devem ser confundidas com funções matemáticas, mas são similares àquelas usadas na programação funcional) simplesmente contêm um conjunto de passos computacionais a serem executados. Um dado procedimento pode ser chamado a qualquer hora durante a execução de um programa, inclusive por outros procedimentos ou por si mesmo. Exemplo: PHP, Visual Basic, Delphi...

Imperativa:
Na Ciência da Computação, programação imperativa é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa. Muito parecido com o comportamento imperativo das linguagens naturais que expressam ordens, programas imperativos são uma sequência de comandos para o computador executar. O nome do paradigma, Imperativo, está ligado ao tempo verbal imperativo, onde o programador diz ao computador: faça isso, depois isso, depois aquilo... Este paradigma de programação se destaca pela simplicidade, uma vez que todo ser humano, ao se programar, o faz imperativamente, baseado na ideia de ações e estados, quase como um programa de computador. Exemplo: Bava, Basic, Cobol, Fortran...

Funcional:
Em ciência da computação, programação funcional é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa. Enfatizando as expressões ao invés de comandos, as expressões são utilizados para cálculo de valores com dados imutáveis. Exemplo: Lisp, Haskel, Elixir, *Dart...

Lógica:
É um paradigma de programação que faz uso da lógica matemática. John McCarthy [1958] foi o primeiro a publicar uma proposta de uso da lógica matemática para programação. A primeira linguagem de programação lógica foi a Planner, a qual permitia a invocação orientada a padrões de planos procedimentais de asserções e de objetivos. Com a necessidade de adaptação aos sistemas de memória muito limitada, que eram disponíveis quando ela foi desenvolvida. A linguagem Planner usava estruturas de controle de backtracking, de tal forma que apenas um único caminho computacional tinha que ser armazenado por vez. Em seguida, o Prolog foi desenvolvido como uma simplificação do Planner que permitia a invocação orientada a padrões apenas a partir de objetivos (também baseado em backtracking).

CLASSE

MOLDE
MODELO

OBJETOS OU INSTÂNCIAS
CASA 1... CASA 2...

ATRIBUTOS
CASA AMARELA
CASA MARROM
QUATIDADE JANELAS

MÉTODOS
ABRIR...
FECHAR...

Aula 3 – Linguagens de Programação

Classes:
As classes de programação são receitas de um objeto, aonde têm características e comportamentos, permitindo assim armazenar propriedades e métodos dentro dela. Para construir uma classe é preciso utilizar o pilar da abstração. Uma classe geralmente representa um substantivo, por exemplo: uma pessoa, um lugar, algo que seja "abstrato".

Uma classe é um tipo definido pelo usuário que contém a "receita", a especificação para os objetos, algo mais ou menos como o tipo inteiro contém o molde para as variáveis declaradas como inteiros. A classe envolve, associa, funções e dados, controlando o acesso a estes, defini-la implica em especificar os seus atributos (dados) e seus métodos (funções).

Um programa que utiliza uma interface controladora de um motor elétrico provavelmente definiria a classe motor. Os atributos desta classe seriam: temperatura, velocidade, tensão aplicada. Estes provavelmente seriam representados na classe por tipos como int ou float. Os métodos desta classe seriam funções para alterar a velocidade, ler a temperatura, etc.

Objetos:
Objetos (computacionais) são caracterizados por atributos e métodos. Atributos são as propriedades de um objeto. Métodos são as ações que um objeto pode realizar. Os objetos são características definidas pelas classes. Neles é permitido instanciar objetos da classe para inicializar os atributos e invocar os métodos.

Atributos:
Atributos são as características de um objeto, essas características também são conhecidas como variáveis, utilizando o exemplo das casas, temos alguns atributos, tais como: cor, quantidadeJanelas, area.

Métodos:
Métodos são as ações que os objetos podem exercer quando solicitados, onde podem interagir e se comunicarem com outros objetos, utilizando o exemplo das casas, temos alguns exemplos: abrirPorta, fecharJanela, acenderLuz.

Construtores
Construtores são basicamente funções de inicialização de uma classe, as quais são invocadas no momento em que objetos desta classe são criadas. Eles permitem inicializar campos internos da classe e alocar recursos que um objeto da classe possa demandar, tais como memória, arquivos, semáforos, soquetes, etc.

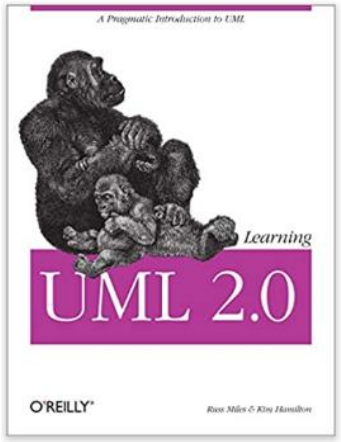
A função construtor de uma classe é reconhecida por ter o mesmo nome da classe. Por exemplo, um construtor para uma classe chamada Rectangle seria a função Rectangle(). É possível definir-se mais de um construtor; em outras palavras, a função membro construtor, assim como qualquer outra função membro de uma classe, pode ser sobrecarregada.

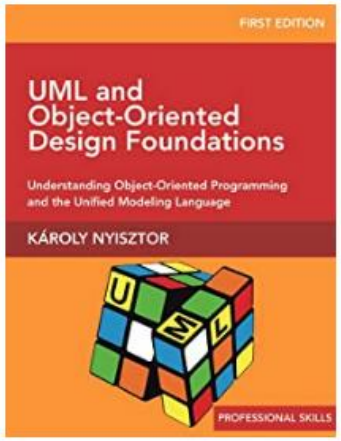
Aula 3 – Linguagens de Programação

Veja ao lado um exemplo de um **diagrama de classe** de um controle de alunos com as seguintes classes, propriedades e métodos.

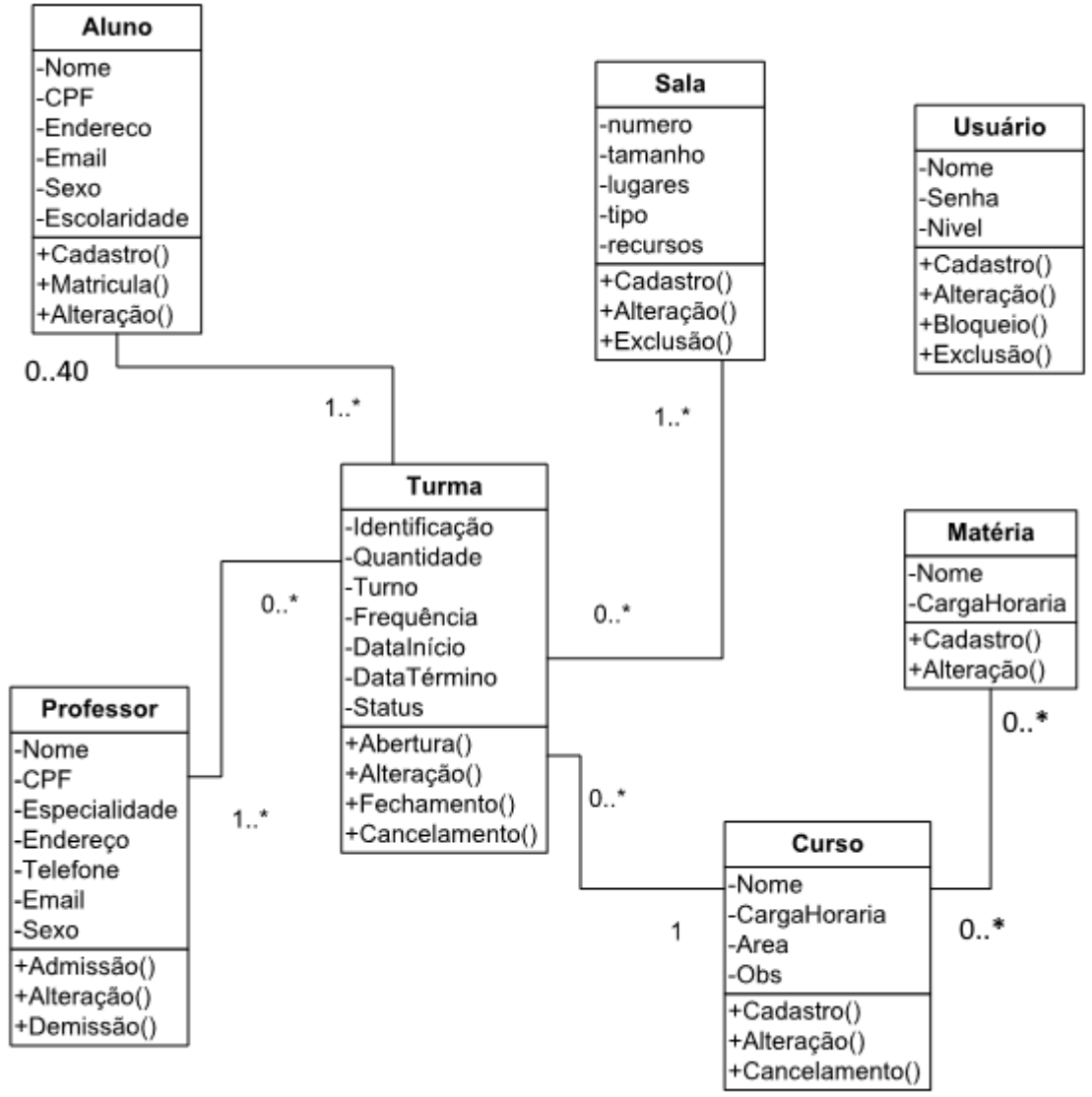
UML, abreviação de Unified Modeling Language, é uma linguagem de modelagem padronizada que consiste em um conjunto integrado de diagramas, desenvolvido para ajudar os desenvolvedores de sistema e software a especificar, visualizar, construir e documentar os artefatos de sistemas de software, bem como para modelagem de negócios e outros sistemas que não sejam de software.

A UML representa uma coleção das melhores práticas de engenharia que provaram ser bem-sucedidas na modelagem de sistemas grandes e complexos. A UML é uma parte muito importante do desenvolvimento de software orientado a objetos e do processo de desenvolvimento de software. A UML usa principalmente notações gráficas para expressar o design de projetos de software.





Softwares para UML
Lucidchart for Enterprise
Visio
Draw.IO
Astah



Aula 3 – Classes: 01-classes.dart

```dart
//Para criar uma classe usamos inical maiúscula
//Em caso de palavra composta, exemplo: CasaDaJane
class Casa {
  //Criando meus atributos
  //Atributos definem minha classe
  String cor = '';
  int quantidadeJanelas = 0;
  int numero = 0;

  //Criando métodos
  //Métodos são ações que pode ser realizadas com nossa classe
  void abrirJanela(String posicao) {
    print('Abrindo a janela...${posicao}');
  }

  void fecharJanela(String posicao) {
    print('Fechando a janela...${posicao}');
  }
}

Run | Debug
void main() {
  //Definido minha classe
  //A casa do john é uma instância de Casa
  //Ou seja uma cópia do modelo original Casa().
  //Assim cadsaDoJohn é um objeto de Casa().
  Casa casaDoJohn = new Casa();

  //Tendo construido um objeto casaDoJohn
  //Podemos colocar valores em seus atributos
  casaDoJohn.cor = 'Marrom';
  casaDoJohn.quantidadeJanelas = 5;
  casaDoJohn.numero = 1970;

  //Acessando o método abrirJanela()
  //É preciso o objeto para fazer uso não só de atributos, mas de métodos também
  casaDoJohn.abrirJanela('Frente!');
  print('A casa do John é: ${casaDoJohn.cor}');
  print('A casa do John tem: ${casaDoJohn.quantidadeJanelas} janelas');
  print('O número da casa do John é: ${casaDoJohn.numero}');
  //Acessando o método fecharJanela()
  casaDoJohn.fecharJanela('Fundos!');

  //A vantagem de utilizar classes é que podemos ter várias instâncias
  //dessa mesma classe
  //então agora posso criar a casa da Jane também, aí ninguém briga :)

  //Criando a instancia da casa da Jane
  Casa casaDaJane = new Casa();

  casaDaJane.cor = 'Verde';
  casaDaJane.quantidadeJanelas = 10;
  casaDaJane.numero = 2000;

  //Saída para os dados da Casa da Jane
  print('------------------------------------------------------------');
  casaDoJohn.abrirJanela('Frente!');
  print('A casa da Jane é: ${casaDaJane.cor}');
  print('A casa da Jane tem: ${casaDaJane.quantidadeJanelas} janelas');
  print('O número da casa da Jane é: ${casaDaJane.numero}');
  casaDoJohn.fecharJanela('Fundos!');
}
```

Aula 3 – Classes: 02-classes.dart

```dart
//Outro exemplo
class Data {
  //Colocamos o LATE para dizer ao dart que o atributo vai ser
  //inicializado futuramente -- NÃO É LEGAL, POIS VOCÊ PODE NÃO INICIÁ-LO
  late int dia;
  late String mes;
  late int ano;
}

Run | Debug
void main() {
  //Criando uma data (3 INSTâNCIAS)
  Data data = new Data();
  Data dataPedido = new Data();
  Data dataSaida = new Data();

  //Inicializando uma data
  data.dia = 3;
  data.mes = 'Janeiro';
  data.ano = 1970;

  //Inicializando a data
  dataPedido.dia = 20;
  dataPedido.mes = 'Março';
  dataPedido.ano = 2015;

  //Inicializando a data
  dataSaida.dia = 29;
  dataSaida.mes = 'Dezembro';
  dataSaida.ano = 2020;

  //Imprimindo as Datas
  print('Mostrar data: ${data.dia}/${data.mes}/${data.ano}');
  print('Mostrar data: ${dataPedido.dia}/${dataPedido.mes}/${dataPedido.ano}');
  print('Mostrar data: ${dataSaida.dia}/${dataSaida.mes}/${dataSaida.ano}');
}
```

01-classes.dart

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Abrindo a janela...Frente!
A casa do John é: Marrom
A casa do John tem: 5 janelas
O número da casa do John é: 1970
Fechando a janela...Fundos!
------------------------------------------------
Abrindo a janela...Frente!
A casa da Jane é: Verde
A casa da Jane tem: 10 janelas
O número da casa da Jane é: 2000
Fechando a janela...Fundos!
Exited
```

02-classes.dart

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Mostrar data: 3/Janeiro/1970
Mostrar data: 20/Março/2015
Mostrar data: 29/Dezembro/2020
Exited
```

Aula 3 – Classes

03-metodos.dart

```dart
class Casa {
  //Atributos definem minha classe
  String cor = '';
  int quantidadeJanelas = 0;
  int numero = 0;

  //Métodos são ações que pode ser realizadas com nossa classe
  void abrirJanela(String posicao) { //Método
    print('Abrindo a janela...${posicao}');
  }

  void fecharJanela(String posicao) { //Método
    print('Fechando a janela...${posicao}');
  }
  //Métodos dentro de métodos
  //This serve para fazer referência a atributos e métodos
  //que estão dentro da mesma classe
  void checarCasa() { //Método
    this.abrirJanela('Frente!');
    this.fecharJanela('Fundos!');
  }
}

Run | Debug
void main() {
  //Criando instância da Classe Casa
  Casa casaDoJohn = new Casa();

  //Atribuindo valores
  casaDoJohn.cor = 'Marrom';
  casaDoJohn.quantidadeJanelas = 5;
  casaDoJohn.numero = 1970;
  //Checando a casa do John
  casaDoJohn.checarCasa();
}
```

04-metodos.dart

```dart
class Data {
  //Colocamos o LATE para dizer ao dart que o atributo vai ser
  //inicializado futuramente -- NÃO É LEGAL, POIS VOCÊ PODE NÃO INICIÁ-LO
  late int dia;
  late String mes;
  late int ano;

  //Criando Métodos
  String exbirData() {
    //Exibir data formatada
    //Essa colocação é inútil, pois não podemos manipular a data
    return '$dia/$mes/$ano';
  }

  //Usando o método toString para retornar a exibição da data
  String toString() {
    //Vai retornar nosso método anterior
    //convertendo em String
    return exbirData();
  }
}

Run | Debug
void main() {
  //Criando uma data (3 INSTâNCIAS)
  Data data = new Data();
  Data dataPedido = new Data();
  Data dataSaida = new Data();

  //Inicializando uma data
  data.dia = 3;
  data.mes = 'Janeiro';
  data.ano = 1970;

```

Aula 3 – Classes

04-metodos.dart (cont)

```dart
  //Inicializando a data
  dataPedido.dia = 20;
  dataPedido.mes = 'Março';
  dataPedido.ano = 2015;

  //Inicializando a data
  dataSaida.dia = 29;
  dataSaida.mes = 'Dezembro';
  dataSaida.ano = 2020;

  /*Imprimindo as Datas
  print('Mostrar data: ${data.dia}/${data.mes}/${data.ano}');
  print('Mostrar data: ${dataPedido.dia}/${dataPedido.mes}/${dataPedido.ano}');
  print('Mostrar data: ${dataSaida.dia}/${dataSaida.mes}/${dataSaida.ano}');
  */

  //Nova Saída
  //Cada uma das instâncias vai receber o omportamento do método exibirData
  //Muito Prático
  String data1 = data.exbirData();
  String data2 = dataPedido.exbirData();
  String data3 = dataSaida.exbirData();

  print('Data atual: $data1');
  print('Data da realização Pedido: $data2');
  print('Data da Saída do Pedido: $data3');

  //Saída pelo método toString
  print('------------------------------------------------');
  //Imprimindo diretamente o objeto!
  print(data);
  print(dataPedido);
  print(dataSaida);
}
```

03-metodos.dart

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Abrindo a janela...Frente!
Fechando a janela...Fundos!
Exited
```

04-metodos.dart

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Data atual: 3/Janeiro/1970
Data da realização Pedido: 20/Março/2015
Data da Saída do Pedido: 29/Dezembro/2020
------------------------------------------------
3/Janeiro/1970
20/Março/2015
29/Dezembro/2020
Exited
```

Aula 3 – Classes: 05-construtor.dart

```dart
class Data {
  //Colocamos o LATE para dizer ao dart que o atributo vai ser
  //inicializado futuramente -- NÃO É LEGAL, POIS VOCÊ PODE NÃO INICIÁ-LO
  late int dia;
  late String mes;
  late int ano;

  //Criando um construtor
  //Se não criar, a linguagem vai criar um padrão
  //Ele tem o mesmo nome da Classe
  //Só posso ter um com o mesmo nome da Classe
  //Colocamos parâmetros para receber os valores
  Data(int diaCorrente, String mesCorrent, int anoCorrent) { //Construtor
    //Atribuimos os valores aos nosso atributos
    dia = diaCorrente;
    mes = mesCorrent;
    ano = anoCorrent;
  }

  String exbirData() { //Método
    return '$dia/$mes/$ano';
  }

  String toString() { //Método
    return exbirData();
  }
}
```

```dart
void main() {
  //Com o construtor criado
  //Posso agora colocar os valores diretamente na criação do objeto
  Data data = new Data(3, 'Janeiro', 1970);
  Data dataPedido = new Data(20, 'Março', 2015);
  Data dataSaida = new Data(29, 'Dezembro', 2020);

  //Cada uma das instâncias vai receber o omportamento do método exibirData()
  //Muito Prático
  String data1 = data.exbirData();
  String data2 = dataPedido.exbirData();
  String data3 = dataSaida.exbirData();

  print('Data atual: $data1');
  print('Data da realização Pedido: $data2');
  print('Data da Saída do Pedido: $data3');

  //Saída pelo método toString
  print('------------------------------------------------');
  //Imprimindo diretamente o objeto!
  print(data);
  print(dataPedido);
  print(dataSaida);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Data atual: 3/Janeiro/1970
Data da realização Pedido: 20/Março/2015
Data da Saída do Pedido: 29/Dezembro/2020
------------------------------------------------
3/Janeiro/1970
20/Março/2015
29/Dezembro/2020
Exited
```

Aula 3 – Classes: 06-construtor_this.dart

```dart
class Data {
  /*Null-safe
  int? variavel1 = 1; //Pode ser null mais tarde
  String? variavel2; //É null no início. Pode também ser null mais tarde!
  */

  int dia; //Retirar o late por conta do Construtor curto
  String mes;
  int ano;

/* CNSTRUTOR FORMA LONGA
  //Criando um construtor
  Data(int dia, String mes, int ano) {
    //Atribuimos os valores aos nosso atributos
    //This representa o objeto da classe atual
    //Assim eu entendo o que é o parâmetro e o que é atributo
    this.dia = dia;
    this.mes = mes;
    this.ano = ano;
  }
  */

  //CONSTRUTOR FORMA CURTA
  //Agora eu posso tirar o late do tipo do atributo
  //Data(this.dia, this.mes, this.ano); //Não esqueça

  //USANDO O CONSTRUTOR COM PARÂMETROS OPCIONAIS
  //Precisamos inicializar os atributos
  Data([this.dia = 0, this.mes = 'vazio', this.ano = 0]); //Não esqueça

  //Criando Métodos
  String exbirData() {
    return '$dia/$mes/$ano';
  }

  //Criando um método para chamar o método exibirData()
  //Para isso vamos usar o método interno toString()
  String toString() {
    //Vai retornar nosso método anterior
    //convertendo em String
    return exbirData();
  }
}

Run | Debug
void main() {
  //Criando uma data (3 INSTâNCIAS)
  Data data = new Data(3, 'Janeiro', 1970);
  Data dataPedido = new Data(20, 'Março', 2015);
  Data dataSaida = new Data(29, 'Dezembro', 2020);
  //Cada uma das instâncias vai receber o omportamento do método exibirData
  //Muito Prático
  String data1 = data.exbirData();
  String data2 = dataPedido.exbirData();
  String data3 = dataSaida.exbirData();
  print('Data atual: $data1');
  print('Data da realização Pedido: $data2');
  print('Data da Saída do Pedido: $data3');

  //Saída pelo método toString
  print('------------------------------------------------');
  //Imprimindo diretamente o objeto!
  print(data);
  print(dataPedido);
  print(dataSaida);
  //Novas datas com com parâmetros opcionais no construtor
  Data dataOpcional = new Data();
  print('------------------------------------------------');
  print(dataOpcional);
  print(
      '${dataOpcional.dia = 10}/${dataOpcional.mes =
      'Outubro'}/${dataOpcional.ano = 2010}');
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
Data atual: 3/Janeiro/1970
Data da realização Pedido: 20/Março/2015
Data da Saída do Pedido: 29/Dezembro/2020
------------------------------------------------
3/Janeiro/1970
20/Março/2015
29/Dezembro/2020
------------------------------------------------
0/vazio/0
10/Outubro/2010
Exited
```

Aula 3 – Classes: 07-construtor_exemplo.dart

```dart
class Retangulo {
  //Atrubutos
  double base;
  double altura;

  //Método Construtor
  //Construtores servem para iniciar os atributos
  //Parâmetros opcionais
  Retangulo([this.base = 1.0, this.altura = 1.0]) {}

  //Método Imprimir valores
  calcularAreaRetangulo() {
    return base * altura;
  }
}

Run | Debug
void main() {
  //Criando um retângulo
  Retangulo retangulo1 = new Retangulo();
  retangulo1.base = 4;
  retangulo1.altura = 10.0;

  double area = retangulo1.calcularAreaRetangulo();
  print('A área do triângulo é: ${area}m²');

  //Saída
}
```

```
DEBUG CONSOLE ···  Filter (e.g
A área do triângulo é: 40.0m²
Exited
```

Aula 3 – Classes: 08-construtor_param_nomeados.dart

```dart
class Data {
  int dia; //Retirar o late por conta do Construtor curto
  String mes;
  int ano;

  //Contrutor Nomeado com Parâmetros nomeados
  Data.nomeada(
      {this.dia = 0, this.mes = 'vazio', this.ano = 0000}); //Não esqueça
  ....
  //Criando Métodos
  String exbirData() {
    return '$dia/$mes/$ano';
  }
  //Criando um método para chamar o método exibirData()
  //Para isso vamos usar o método interno toString()
  String toString() {
    //Vai retornar nosso método anterior
    //convertendo em String
    return exbirData();
  }
}
Run | Debug
void main() {
  //Instanciando o objeto dataDia
  print(new Data.nomeada());
  print('Dia: ${Data.nomeada(dia: 10)}');
  print('mês: ${Data.nomeada(mes: 'Outubro')}');
  print('Ano: ${Data.nomeada(ano: 2000)}');
  print('Data completa: ${Data.nomeada(ano: 2000, mes: 'Outubro', dia: 10)}');
  print('//////////////Atribuição a uma variável//////////////////');
  //Atribuindo os valores a uma variável
  var dataPagamento = Data.nomeada(dia: 1, mes: 'Janeiro', ano: 2021);

  //Saída
  print(dataPagamento);
}
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  ...  Filter (e.g. text, !exclude)
0/vazio/0
Dia: 10/vazio/0
mês: 0/Outubro/0
Ano: 0/vazio/2000
Data completa: 10/Outubro/2000
//////////////Atribuição a uma variável//////////////////
1/Janeiro/2021
Exited
```

Aula 3 – Classes: 09-import.dart / classes.dart (dentro da pasta model)

```dart
//Criar uma pasta chamada model

//Fora do diretório corrente
//import '../model/classes.dart';

//no diretorio corrente
import 'model/classes.dart';

Run | Debug
void main() {

  //Criando uma conta
  Banco conta = new Banco('John Doe', 123456);

  conta.imprimeUsuario();

}
```

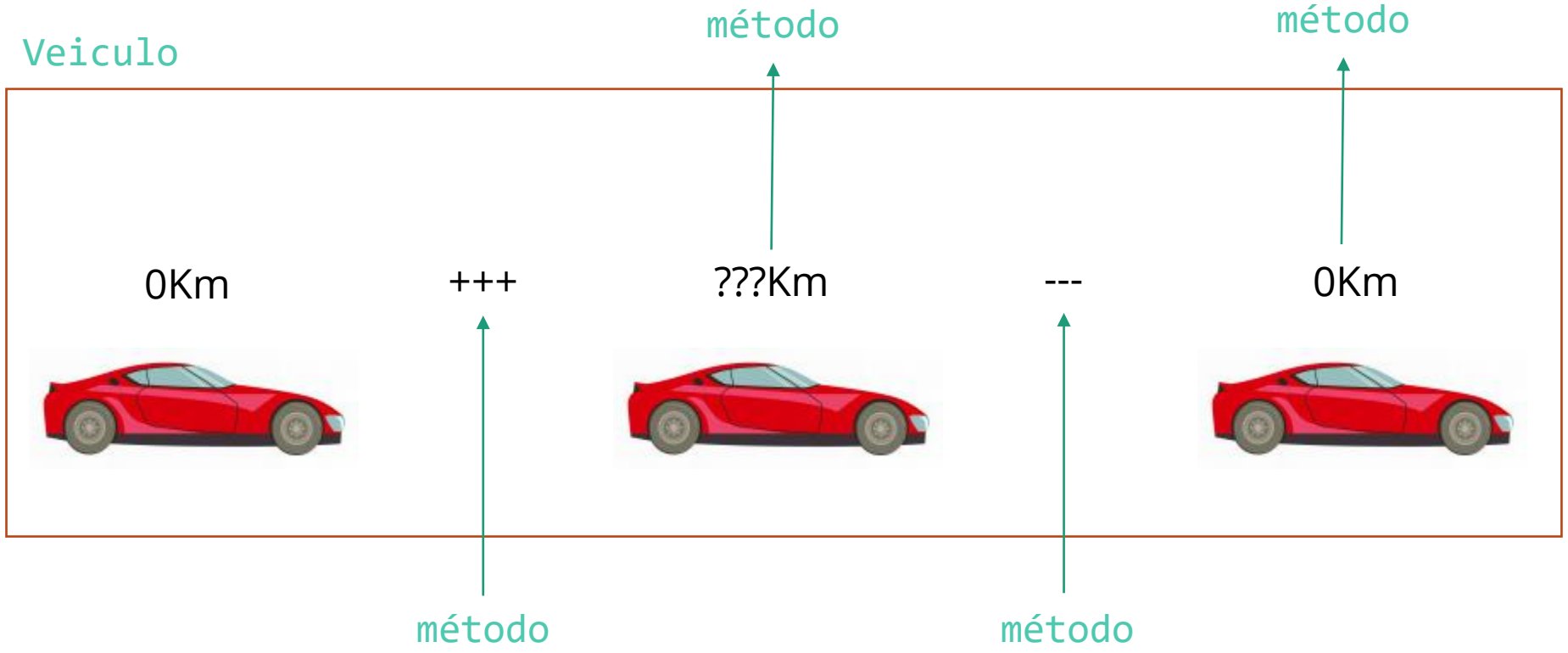
```dart
class Banco {
  //Atrubutos
  String nome;
  int conta;

  //Método Construtor
  Banco(this.nome, this.conta) {}

  //Método Imprimir usuário
  imprimeUsuario() {
    print(this.nome);
    print(this.conta);
  }
}
```

```
John Doe
123456
Exited
```

Aula 3 – Atividade 1 - OO

Crie um programa em Dart utilizando o método de importação de classes. Para esse programa suponha que um veículo acelera até um limite de velocidade pré-estabelecido. Uma verificação será feita quando ele chegar nesse limite e assim começará a desacelerar até parar. O programa deverá mostrar as etapas da aceleração e desaceleração.

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   ...
Acelerando o Camaro: 10Km/h
Acelerando o Camaro: 20Km/h
Acelerando o Camaro: 30Km/h
Acelerando o Camaro: 40Km/h
Acelerando o Camaro: 50Km/h
Acelerando o Camaro: 60Km/h
Acelerando o Camaro: 70Km/h
Acelerando o Camaro: 80Km/h
Acelerando o Camaro: 90Km/h
Acelerando o Camaro: 100Km/h
O camaro chegou a 100Km/h
Freando o Camaro: 90Km/h
Freando o Camaro: 80Km/h
Freando o Camaro: 70Km/h
Freando o Camaro: 60Km/h
Freando o Camaro: 50Km/h
Freando o Camaro: 40Km/h
Freando o Camaro: 30Km/h
Freando o Camaro: 20Km/h
Freando o Camaro: 10Km/h
Freando o Camaro: 0Km/h
O Camaro parou a 0Km/h
Exited
```



Aula 03 – Visibilidade: Getters and Setters – 11-getter_setter.dart

Getters e setters são métodos especiais que fornecem acesso de leitura e gravação às propriedades(atributos) de um objeto. Cada variável de instância de sua classe tem um getter implícito e um setter, se necessário. No Dart, você pode levar isso ainda mais longe implementando seus próprios getters e setters.

- Os getters são definidos usando a palavra-chave get **sem parâmetros e retorna um valor**.
- Os setters são definidos usando a palavra-chave set **com um parâmetro e nenhum valor de retorno**.

```dart
//get serve para pegar alguma coisa
//set serve para confirugar alguma coisa

class Conta {
  //Atributos
  //o (_) underline significa Privado (VISIBILIDADE)
  //Não pode ser acesado diretamente pelo meu objeto conta
  //ou seja, só poderá ser utilizado dentro da classe
  //1º PILAR ENCAPSULAMENTO
  double _saque;
  double saldo;

  Conta([this._saque = 100, this.saldo = 1000.0]); //Construtor

  //Como o atributo não pode ser acessado diretamente
  //Vamos usar get e set para configurá-lo
  //Fazendo isso não acessamos diretamente nosso atributos
  //Atenção: Só coloque essas configurações se realmente precisar
  //proteger seu atributo

  //get -- pegando o valor
  //get não possui parâmetro e retorna valor
  double get getSaque {
    return this._saque;
    //return 400.0;
  }

  //set -- configurando o valor
  //set possui parâmetro mas não retorna valor
  set setSaque(double saque) {
    //Poso aqui vazer verificações e validações
    if (saque > 0 && saque <= saldo) {
      this._saque = saque;
    } else {
      print('Valor indisponível!');
    }
  }
}

Run | Debug
void main() {
  //Criando um conta
  Conta conta = new Conta();

  //Com o valor de saque setado (set)
  //Posso atribuir valor ao saque
  //_saque fica sem alteração, ou seja, valor 0
  //não precisa de parenteses
  conta.setSaque = 0;

  //Imprimindo
  print('Valor sacado: ${conta.getSaque}');
}
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE
Valor indisponível!
Valor sacado: 100.0
Exited
```

Aula 03 – Visibilidade: Getters and Setters – 12-getter_setter.dart

```dart
class Pessoa {
  String nome;
  String _cpf; //privado
  double peso;
  double _altura; //privado
  //Construtor
  Pessoa(
      [this.nome = '',
      this._cpf = '000.000.000-00',
      this.peso = 0.0,
      this._altura = 0.0]);
  //Métodos Especiais
  String get getCpf {
    return _cpf;
  }
  set setCpf(String cpf) {
    if (cpf.length != 14) {
      print('CPF Inválido');
      print('---------------------------------');
    } else {
      this._cpf = cpf;
    }
  }
  double get getAltura {
    return _altura;
  }
  set setAltura(double altura) {
    if (altura < 0 || altura > 2.50) {
      print('Altura inválida');
      print('---------------------------------');
    } else {
      this._altura = altura;
    }
  }
}

```

```dart
void main() {
  Pessoa paciente = new Pessoa();
  paciente.nome = 'John Doe';
  paciente.setCpf = '123.456.789-12';
  paciente.setAltura = 1;
  paciente.peso = 80.0;

  print('Nome do paciente: ${paciente.nome}');
  print('CPF do paciente: ${paciente.getCpf}');
  print('Altura do paciente: ${paciente.getAltura}');
  print('Peso do paciente: ${paciente.peso}');
}
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE
Nome do paciente: John Doe
CPF do paciente: 123.456.789-12
Altura do paciente: 1.0
Peso do paciente: 80.0
Exited
```

Aula 03 – Herança 13-heranca.dart

```dart
//Definindo duas classes distintas

class Cachorro {
  String cor;
  double peso;
  String raca;

  //Construtor
  Cachorro({required this.cor, required this.peso, required this.raca});
}

class Gato {
  String cor;
  double peso;
  String raca;

  //Construtor
  Gato({required this.cor, required this.peso, required this.raca});
}
```

Aula 03 – Herança 14-heranca.dart

```dart
//Agregando Métodos as nossas classes
class Cachorro {
  String cor;
  double peso;
  String raca;

  //Construtor
  Cachorro({required this.cor, required this.peso, required this.raca});

  //Método
  void latir() {
    print('O cachorro está latindo...');
  }
  void farejar() {
    print('O cachorro está farejando...');
  }
}

class Gato {
  String cor;
  double peso;
  String raca;

  //Construtor
  Gato({required this.cor, required this.peso, required this.raca});

  //Método
  void miar() {
    print('O cachorro está latindo...');
  }
  void farejar() {
    print('O cachorro está farejando');
  }
}
```

Aula 03 – Herança 15-heranca.dart

```dart
//Herança
/**
 * Herança é um mecanismo que permite que características comuns a
 * diversas classes sejam fatoradas em uma classe base, ou
 * superclasse. A partir de uma classe base, outras classes
 * podem ser especificadas.
 */

//Classe Pai
class Animal {
  String cor;
  double peso;
  String raca;

  //Construtor
  Animal({this.cor = '', this.peso = 0.0, this.raca = ''});
}

//Para a relação de herança utilizamos a palava extends
class Cachorro extends Animal {
  //Método específico do Cachorro
  void latir() {
    print('O animal está latindo...');
  }
}

class Gato extends Animal {
  //Método específico do Gato
  void miar() {
    print('O animal está miando');
  }
}

```

```dart
void main() {
  Cachorro toto = new Cachorro();
  Gato felix = new Gato();

  //Definindo os valores dos atributos
  toto.cor = 'Marrom';
  felix.peso = 3.5;

  print('A cor do cachorro é: ${toto.cor}');
  toto.latir();
  print('A peso do gato é: ${felix.peso}');
  felix.miar();
}
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE
A cor do cachorro é: Marrom
O animal está latindo...
A peso do gato é: 3.5
O animal está miando
Exited
```

Aula 03 – Herança 16-heranca.dart

```dart
//Sobrescrita de métodos
//Utilizando exemplo anterior nós poderíamos ter um método
//compartilhado com os dois animais
//por exemplo: farejar()

//Classe Pai
class Animal {
  String cor;
  double peso;
  String raca;

  //Construtor
  Animal({this.cor = '', this.peso = 0.0, this.raca = ''});

  //Método comum para os dois animais
  void farejar() {
    print('O animal está farejando com um ');
  }
}

class Cachorro extends Animal {
  //Método específico do Cachorro
  void latir() {
    print('O animal está latindo...');
  }

  //Sobrescrevendo
  @override
  void farejar() {
    super.farejar();
    print('Cachorro');
  }
}

class Gato extends Animal {
  //Método específico do Gato
  void miar() {
    print('O animal está miando');
  }

  //Sobrescrevendo
  @override
  void farejar() {
    super.farejar();
    print('Gato');
  }
}

Run | Debug
void main() {
  Cachorro toto = new Cachorro();
  Gato felix = new Gato();

  //Definindo os valores dos atributos
  toto.cor = 'Marrom';
  felix.peso = 3.5;

  print('A cor do cachorro é: ${toto.cor}');
  toto.latir();
  print('A peso do gato é: ${felix.peso}');
  felix.miar();
  print('--------------------------------');
  toto.farejar();
  felix.farejar();
  print('--------------------------------');

}
```

```
PROBLEMS 2   OUTPUT   DEBUG CONSOLE
A cor do cachorro é: Marrom
O animal está latindo...
A peso do gato é: 3.5
O animal está miando
--------------------------------
O animal está farejando com um
Cachorro
O animal está farejando com um
Gato
--------------------------------
Exited
```

Siga o Senac em Minas nas Redes Sociais: