



# Introdução ao Flutter e Dart



Prof<sup>a</sup>. Ivre Marjorie  
(ivre@pucminas.br)

---

# Visão Geral do Flutter

---



# O que é Flutter?



- É um framework construído pelo Google para o desenvolvimento de multiplataforma
- Como o mesmo **código base**, gera aplicativos **nativos** para Android, IOS, Web e Desktop
- Utiliza o Dart como linguagem de desenvolvimento

# Porque Flutter?

- Criado e mantido pelo Google
- Mais de 170 widgets (componentes) prontos para serem utilizados
- Principais plugings para acesso à recursos nativos no celular também são mantidos pelo Google

# Porque Flutter?

- Seu código Dart é executado diretamente pelo aparelho
- Flutter desenha todos os pixels na tela, tornando o aplicativo altamente customizável
- Alta performance: aplicativos rodam em 60 frames por segundo (ou até 120 caso o aparelho suporte)

# Porque Flutter?

- Alta produtividade: alterações no código refletidos no celular ou emulador em até 0,5 s
- Baseado em **três** pilares:
  1. Desenvolvimento rápido
  2. Interfaces bonitas
  3. Performance nativa

# Porque Flutter?

- Alta produtividade: alterações no código refletidos no celular ou emulador em até 0,5 s
- Baseado em **três** pilares:
  1. Desenvolvimento rápido
  2. Interfaces bonitas
  3. Performance nativa

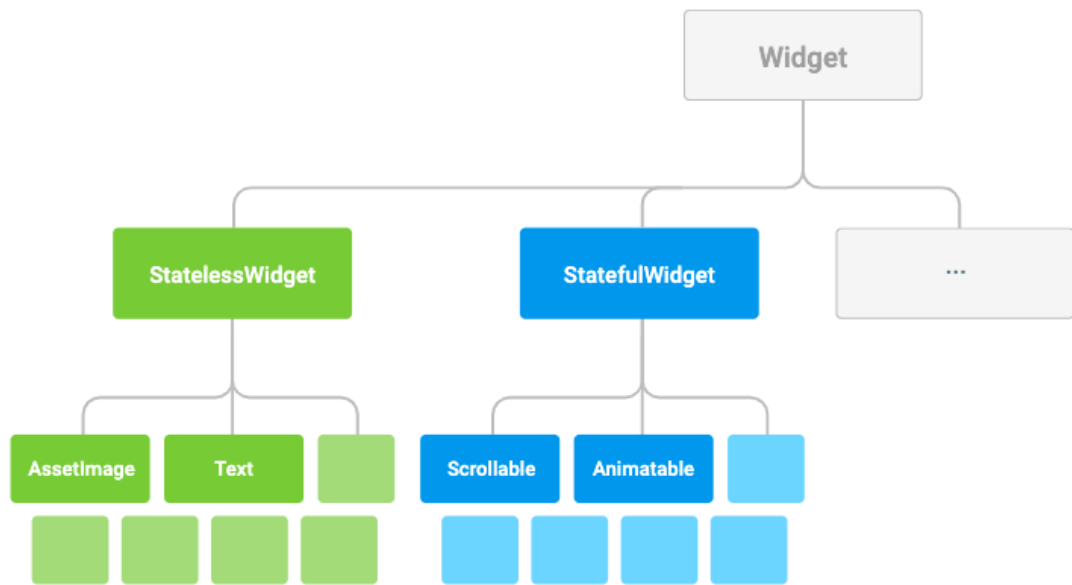
# Quem está usando o Flutter?





# Flutter - Widgets

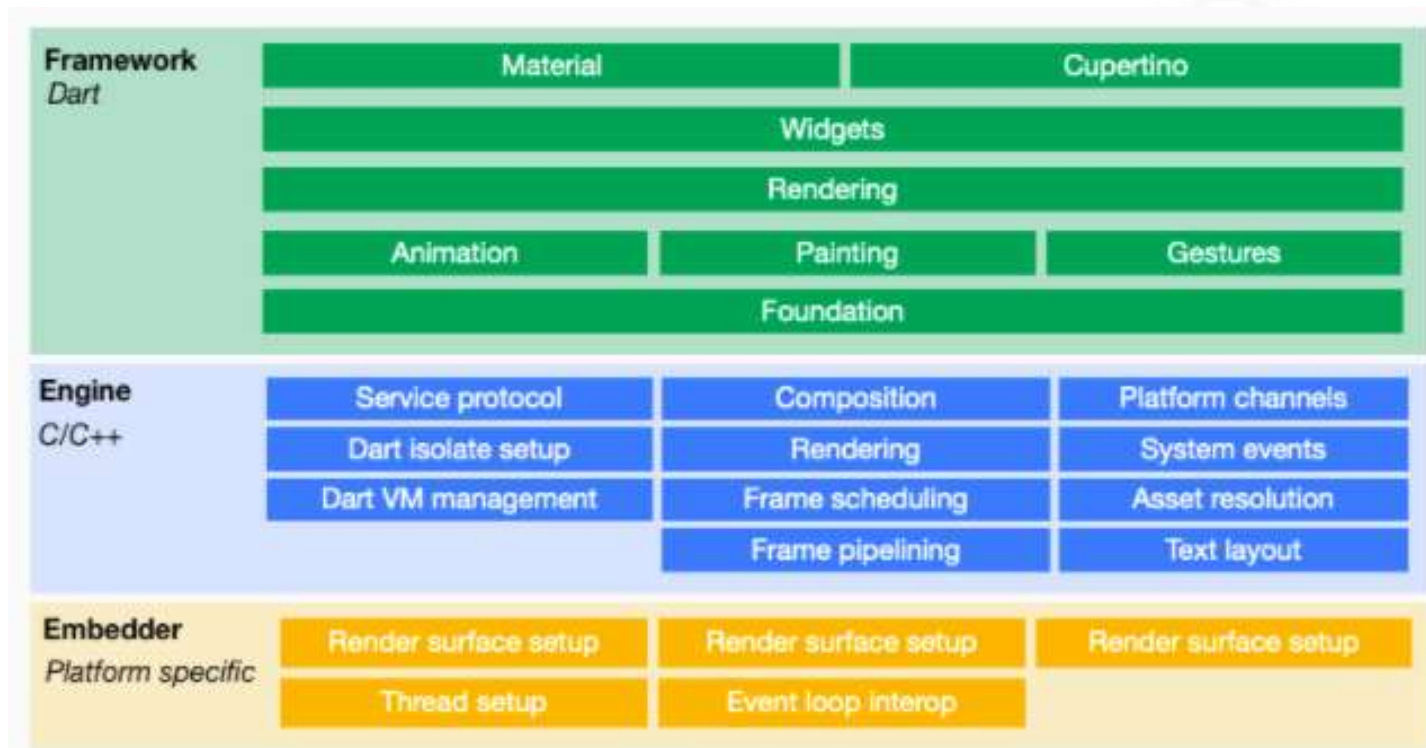
No Flutter, tudo é considerado um Widget, desde um tema para sua aplicação, como também um componente mais complexo



# Flutter - Widgets

- Diferente de outras estruturas que separam visualizações, controladores de exibição, layouts e outras propriedades, o Flutter possui um modelo de objeto unificado e consistente: o **widget**

# Flutter - Arquitetura



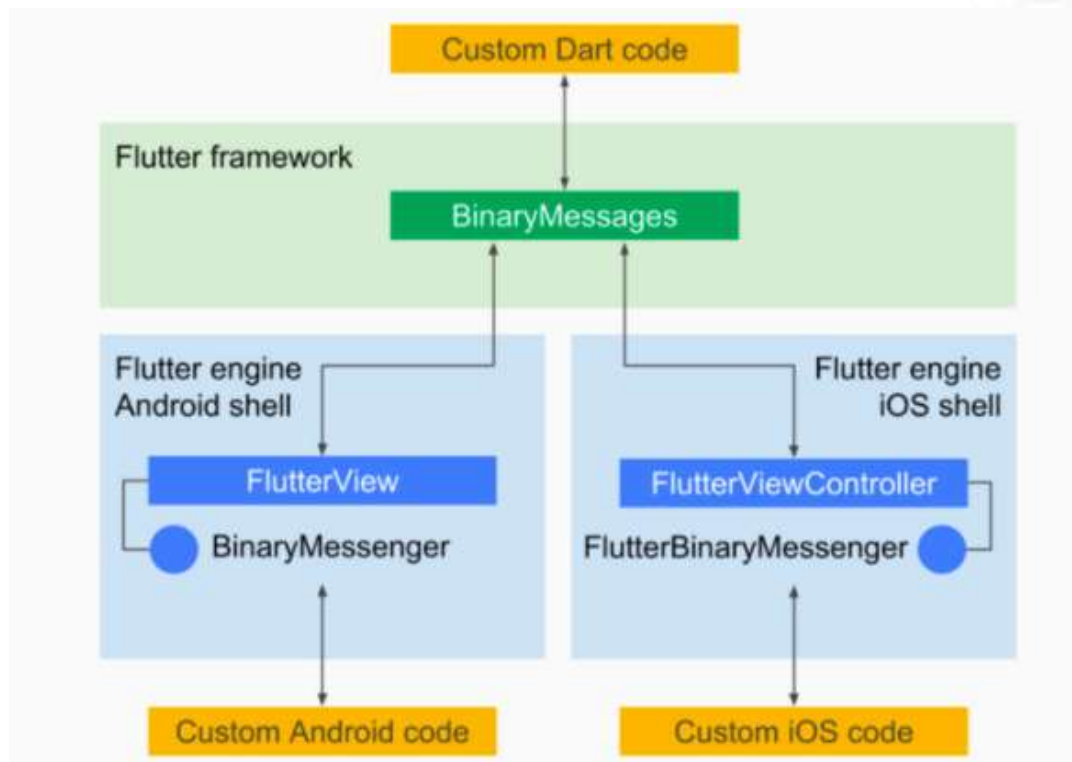
# Flutter - Arquitetura

- A primeira camada é responsável por todo o Framework Flutter escrito em Dart, onde vamos trabalhar
- A segunda camada se trata do core do Flutter, lá são feitos tratamentos específicos de cada sistema e a engine gráfica

# Flutter - Arquitetura

- A terceira camada é responsável por incorporar o Flutter em várias plataformas, suas principais tarefas são: renderizar configurações de superfície, configurações de thread e plug-ins. Ou seja, facilitar a comunicação com a plataforma Android e IOS.

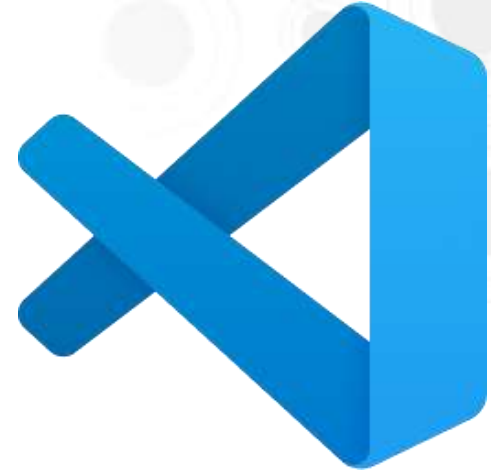
# Flutter - Arquitetura



# Ferramentas de Desenvolvimento



**Android  
Studio**



**Visual Studio Code**

---

# Introdução ao Dart

---





# O que é Dart?



- Linguagem de programação criada pela Google em 2011
- A ideia inicial da criação do Dart, era substituir o Javascript, mas isso não vingou
- O Google mantém tanto o Flutter, quanto o Dart

# O que é Dart?



- O Flutter é a “ferramenta” que utiliza essa linguagem, tornando possível o desenvolvimento de aplicações para dispositivos móveis
- Sendo assim, o Dart funciona sem o Flutter, mas o Flutter não funciona sem o Dart

# O que é Dart?



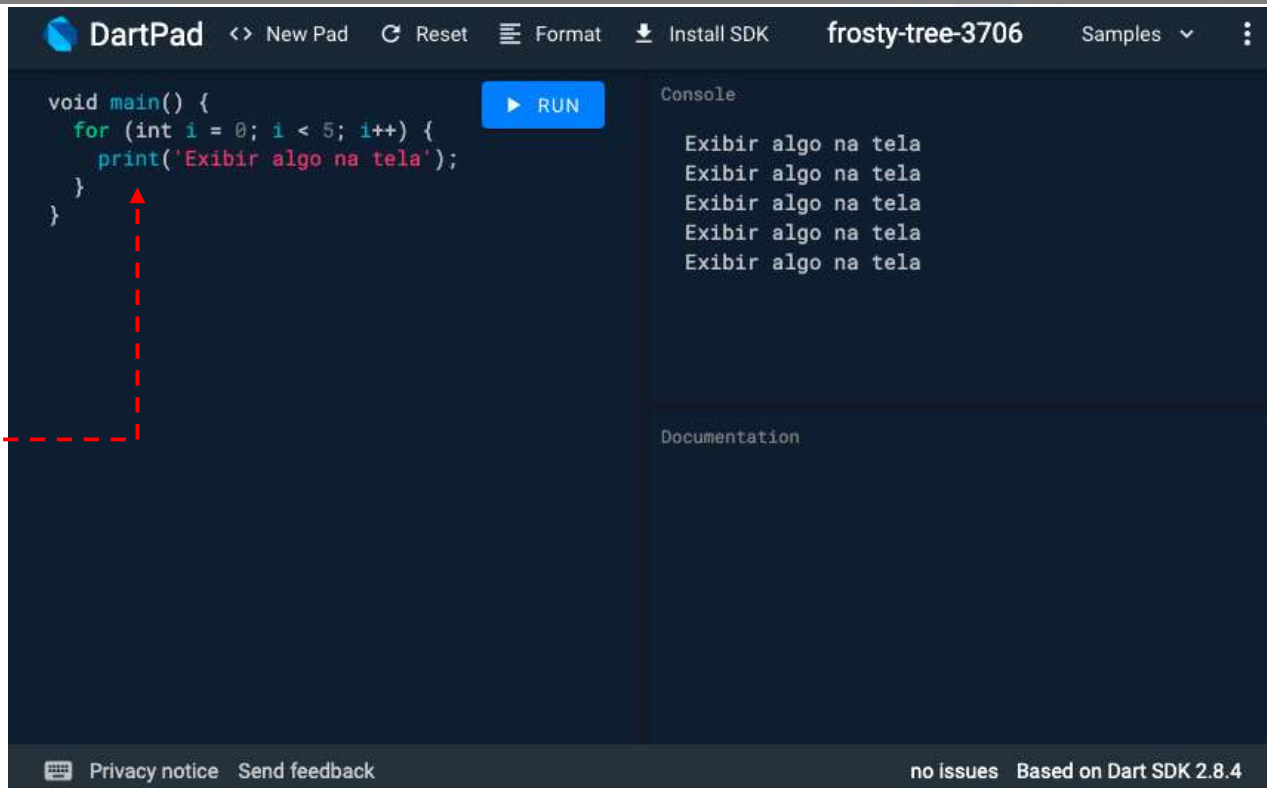
- Foi construído desde o início pensando em desenvolvimento rápido e multiplataformas
- Apesar de ser outra linguagem, é muito simples e similar a outras linguagens, tais como: C, C#, Java e Javascript

# DartPad

- No site oficial da linguagem Dart, temos um editor e compilador de códigos online
- Como o objetivo dessa aula é aprender um pouco da linguagem, vamos usar essa ferramenta
- Para acessar o DartPad: <https://dartpad.dartlang.org/>

# DartPad

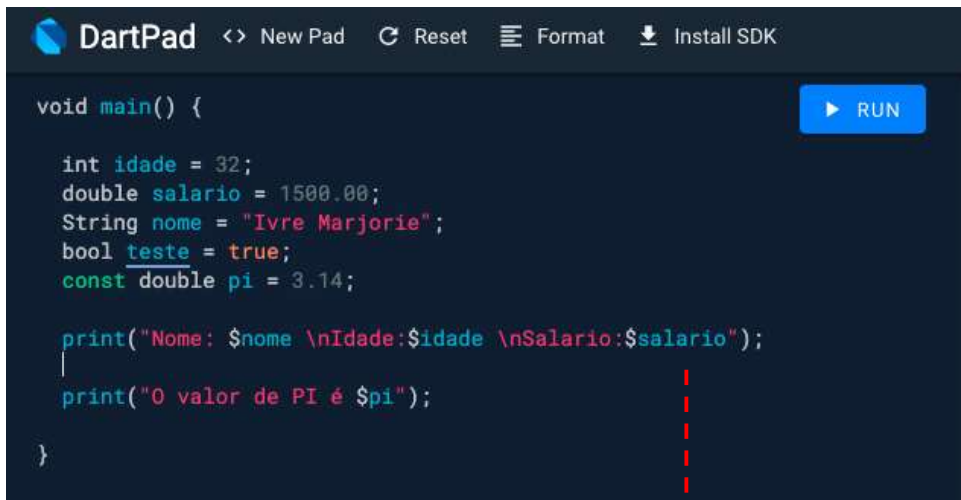
Comando  
**print();** para  
imprimir na  
tela



# Dart Básico: Variáveis

- Espaço na memória capaz de armazenar um valor
- Os principais tipos de variáveis em Dart: String, int, double, bool, var

# Dart Básico: Variáveis



The screenshot shows the DartPad web interface. At the top, there's a toolbar with icons for 'New Pad', 'Reset', 'Format', and 'Install SDK'. Below the toolbar, the code editor contains the following Dart code:

```
void main() {  
  
  int idade = 32;  
  double salario = 1500.00;  
  String nome = "Ivre Marjorie";  
  bool teste = true;  
  const double pi = 3.14;  
  
  print("Nome: $nome \nIdade:$idade \nSalario:$salario");  
  |  
  print("O valor de PI é $pi");  
  
}
```

A blue 'RUN' button is located to the right of the code editor. A red dashed arrow points from the first `print` statement to a text box below.

Imprimindo na tela o valor da variável, usar \$ antes do nome da variável



The screenshot shows the console output of the Dart code. The text is as follows:

```
Console  
  
Nome: Ivre Marjorie  
Idade:32  
Salario:1500  
O valor de PI é 3.14
```

# Dart Básico: Variáveis (Var e Dynamic)

- **var:** uma variável desse tipo pode receber qualquer tipo de dado, o tipo é inferido pelo compilador a partir do primeiro valor atribuído a variável
- **dynamic:** uma variável desse tipo pode assumir tipos dinamicamente, sendo possível alterar durante a execução



# Dart Básico: Variáveis (Var e Dynamic)

```
void main() {  
    dynamic idade = 15;  
  
    print("A idade da Joana é: $idade anos");  
  
    -- idade = "dez";  
  
    print("A idade da Joana é: $idade anos");  
}
```

Console

```
A idade da Joana é: 15 anos  
A idade da Joana é: dez anos
```

Observe que a variável idade começou recebendo um valor inteiro (10) e depois uma string “dez”

# Dart Básico: Operadores de Atribuição

**=** Atribui um valor

**+=** utilizado para somar um valor ao valor já existente na variável

**-=** utilizado para subtrair um valor ao valor já existente na variável

Dessa mesma forma, temos: **\*=** e **/=**

# Dart Básico: Operadores Aritméticos

+ (adição)

- (subtração)

\* (multiplicação)

/ (divisão)

% (resto da divisão)

# Dart Básico: Funções Matemáticas

Método	Descrição
<code>abs()</code>	Retorna o valor absoluto do número.
<code>ceil()</code>	Retorna o último inteiro imediatamente superior.
<code>ceilToDouble()</code>	Retorna o último número imediatamente superior com o tipo <code>double</code> .
<code>clamp(num min, num max)</code>	Se o número estiver dentro do limite, retorna o número. Se não, retorna o limite o qual ele extrapolou.
<code>compareTo(num outro)</code>	Compara com outro número, retornando 1 quando forem diferentes e 0 quando forem iguais.
<code>floor()</code>	Funções matemáticas
<code>floorToDouble()</code>	Arredonda o número para o número inteiro anterior no tipo <code>double</code> .
<code>remainder(num outro)</code>	Retorna a sobra da divisão com outro número.
<code>round()</code>	Arredonda o número para o inteiro mais próximo.
<code>roundToDouble()</code>	Arredonda o número para o valor inteiro mais próximo no tipo <code>double</code> .

# Dart Básico: Funções Matemáticas

<code>toDouble()</code>	Converte o número para Double.
<code>toInt()</code>	Converte o número para Int.
<code>toString()</code>	Converte o número em uma String.
<code>toStringAsExponential([int digitos])</code>	Converte para string com exponencial.
<code>toStringAsFixed(int decimais)</code>	Converte para String contendo N casas decimais.
<code>toStringAsPrecision(int digitos)</code>	Converte para String contendo N dígitos.
<code>truncate()</code>	Retira as casas decimais, retornando um inteiro.
<code>truncateToDouble()</code>	Retira as casas decimais, retornando um double.

# Dart Básico: Operadores Relacionais

Operação	Descrição	Exemplo
==	Igual	k == 2
!=	Diferente	k != 12
>	Maior que	k > 67
<	Menor que	k < 12
>=	Maior ou Igual a	k >= 45
<=	Menor ou Igual a	k <= 23

# Dart Básico: Operadores Lógicos

Tabela <b>E</b> <b>&amp;&amp;</b>	Tabela <b>OU</b> <b>  </b>	Tabela <b>NÃO</b> <b>!</b>
<b>V e V = V</b>	<b>V ou V = V</b>	<b>Não V = F</b>
<b>V e F = F</b>	<b>V ou F = V</b>	<b>Não F = V</b>
<b>F e V = F</b>	<b>F ou V = V</b>	
<b>F e F = F</b>	<b>F ou F = F</b>	

# Dart Básico: Estrutura Condicional

Para incluir condições em Dart, podemos usar os seguintes comandos: **if**, **else**, **else if**, **switch/case**

```
void main() {  
  
    double nota1, nota2, nota3, media;  
    nota1 = 10;  
    nota2 = 5;  
    nota3 = 6;  
    media = (nota1 + nota2 + nota3)/3.0;  
  
    if(media>6)  
        print("Aprovado com media: $media");  
    else if(media >6 && media <=4)  
        print("Em exame especial com media: $media");  
    else  
        print("Reprovado com media: $media");  
  
}
```



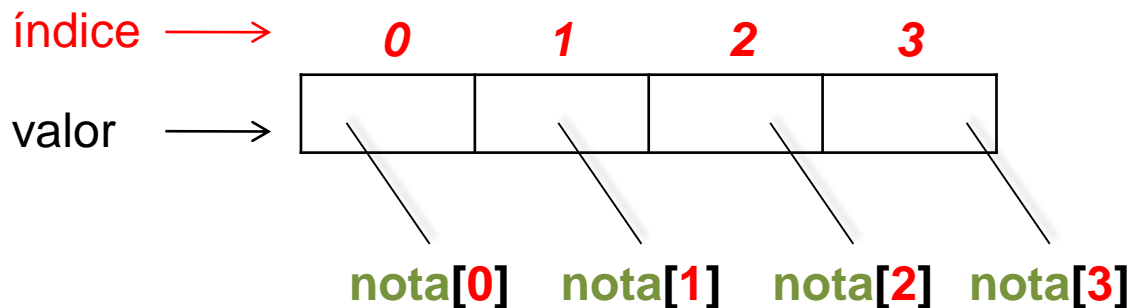
# Dart Básico: Estrutura de Repetição

Para realizar repetições em Dart, podemos usar os seguintes comandos: **for**, **while**, **do-while** ou **forEach**

```
void main() {  
  
  int i = 0, num, soma = 0;  
  
  while(i < 5)  
  {  
    num = i;  
    soma += num;  
    i++;  
  }  
  
  print("A soma é: $soma");  
}
```

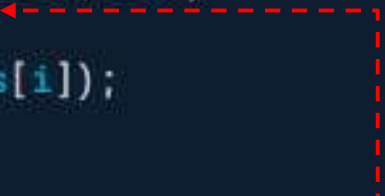
# Dart Básico: Arrays

- **Array:** variável capaz de armazenar mais de um valor, para isso, é necessário o uso de índices para acessar cada uma das posições de um array



# Dart Básico: Arrays

```
void main() {  
  var nomes = ["Raquel", "Roberto", "Pietra"];  
  
  print("Nomes do array:");  
  for(int i=0; i<3; i++)  
  {  
    print(nomes[i]);  
  }  
}
```



Lembrando que o índice do array começa em zero, por isso, a variável *i* começa com o valor zero

Console

Nomes:  
Raquel  
Roberto  
Pietra

# Dart Básico: Funções

- **Função:** é um conjunto de instruções desenhadas para cumprir uma tarefa particular e agrupadas numa unidade com um nome para referenciá-las
- Para criar um função:

```
TipoFunção NomeFunção ( argumentos )  
{  
    //comandos  
}
```

# Dart Básico: Funções

- Para que as funções sejam executadas, é necessário realizar a **chamada da função**, para isso, basta incluir o nome da função e os parênteses, passando ou não os parâmetros

# Dart Básico: Funções

```
void calculaMedia(double n1, double n2, double n3)
{
    double media;
    media = (n1+n2+n3)/3.0;
    print("A media das notas é: $media");
}

void main() {
    double nota1, nota2, nota3;
    nota1=10;
    nota2=5;
    nota3=6;

    calculaMedia(nota1,nota2,nota3);
}
```

Função  
CalculaMedia(),  
que recebe 3  
parâmetros

Chamada da  
função  
CalculaMedia()

# Dart Básico: Funções

```
double calculaMedia(double n1, double n2, double n3)
{
    double media;
    media = (n1+n2+n3)/3.0;
    return media;
}
```

Função  
CalculaMedia()  
agora retorna o  
valor calculado

```
void main() {
    double nota1, nota2, nota3, media;
    nota1=10;
    nota2=5;
    nota3=6;
    media = calculaMedia(nota1,nota2,nota3);

    print("A media é: $media");
}
```

Chamada da  
função  
CalculaMedia()

# Dart Intermediário: Classes e Objetos

- Uma **classe** é a representação de um conjunto de objetos que compartilham a mesma estrutura de atributos, operações e relacionamentos dentro de um mesmo contexto
- Cada **objeto** individual é então criado com base no que está definido na classe



# Dart Intermediário: Classes e Objetos

- Exemplo de um objeto (instância) da classe **Pessoa**:

## **Pessoa**

```
String nome;  
int idade;  
double altura;  
void CadastrarDados(){}  
void MostrarDados(){}  
void Aniversario(){}  

```

## **Obj – Maria Clara**

```
nome = Maria Clara  
idade = 4 anos  
altura = 1.05
```

# Dart Intermediário: Classes e Objetos

```
class Pessoa{  
  //Atributos  
  String nome;  
  int idade;  
  double altura;  
  //Métodos  
  void CadastrarDados(){  
    nome = "Maria Clara";  
    idade = 4;  
    altura = 1.05;  
  }  
  void MostrarDados(){  
    print("Nome: $nome \nIdade: $idade \nAltura: $altura");  
  }  
  void Aniversario(){  
    idade++;  
  }  
}  
  
void main() {  
  Pessoa obj = new Pessoa();  
  obj.CadastrarDados();  
  obj.Aniversario();  
  obj.MostrarDados();  
}
```

▶ RUN

Console

```
Nome: Maria Clara  
Idade: 5  
Altura: 1.05
```

# Dart Intermediário: Construtores

- A inicialização automática de um objeto é obtida pela chamada a um método especial
- Esse método é conhecido como **construtor** e executado quando o objeto é criado na memória
- O **construtor** é um método que tem o mesmo nome da classe e é executado automaticamente toda vez que um objeto é criado

# Dart Intermediário: Construtores

```
class Pessoa{  
  //Atributos  
  String nome;  
  int idade;  
  double altura;  
  //Métodos  
  Pessoa(){  
    nome = "Maria Clara";  
    idade = 4;  
    altura = 1.05;  
  }  
  void MostrarDados(){  
    print("Nome: $nome \nIdade: $idade \nAltura: $altura");  
  }  
  void Aniversario(){  
    idade++;  
  }  
}  
  
void main() {  
  Pessoa obj = new Pessoa();  
  obj.MostrarDados();  
}
```

▶ RUN

Console

```
Nome: Maria Clara  
Idade: 4  
Altura: 1.05
```

Construtor método  
com o mesmo nome  
da classe

O construtor é executado  
automaticamente, quando  
o objeto é instanciado

# Dart Intermediário: Getter e Setter

- **Getter:** usado para obter algo
- **Setter:** usado para configurar

# Dart Intermediário: Getter e Setter

```
class Conta{  
  double saldo = 0;  
  double _saque = 0;  
  //Getter  
  double get saque{  
    return this._saque;  
  }  
  //Setter  
  set saque(double saque){  
    if(saque >0 && saque <=500){  
      this._saque = saque;  
    }  
  }  
}
```

```
void main() {  
  Conta conta1 = new Conta();  
  Conta conta2 = new Conta();  
  conta1.saque = 500; //setter  
  print(conta1.saque); //getter  
  conta2.saque = 900;  
  print(conta2.saque);  
}
```

Console

500

0

No setter colocamos uma validação de uma condição, para realizar um saque o valor de no máximo R\$ 500

# Dart Intermediário: Herança

- É possível verificar que algumas classes possuem atributos e métodos semelhantes a outras classes
- Em vez, de criarmos classes com atributos e/ou métodos semelhantes, usamos a relação de **herança**
- É possível uma classe herdar de outra classe atributos e/ou métodos e ter diferenças que só dizem respeito a essa classe

# Dart Intermediário: Herança

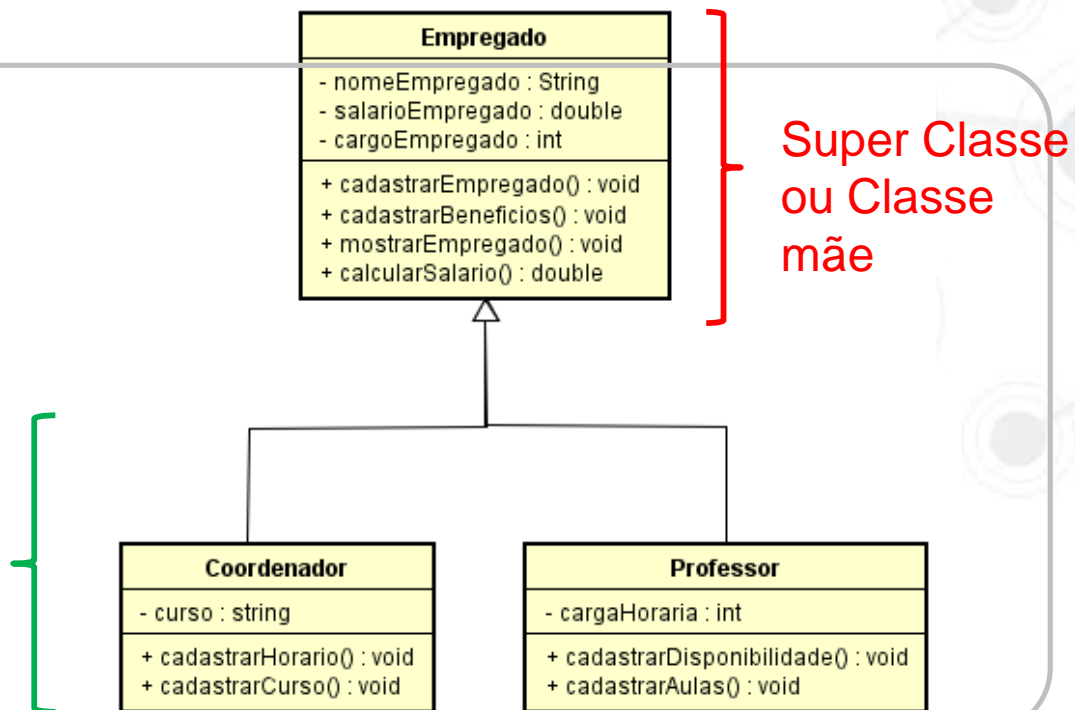
- O processo de herança de classes permite criar uma classe que herda as características de outra classe ou outras classes já existentes
- A herança oferece um meio de relacionar classes umas com as outras por intermédio de hierarquias



# Dart Intermediário: Herança

- Exemplo de herança:

Sub Classe  
ou Classes  
filhas



# Dart Intermediário: Herança

```
class Empregado{
    String nome;
    int matricula;
    double salario;
}

class Professor extends Empregado{
    int horasAula;
    Professor(int horas){
        horasAula = horas;
    }
    void CalculaSalario(){
        salario = 120.0 * horasAula;
    }
}

class Coordenador extends Empregado{
    void CalculaSalario(){
        salario = 250.0 * 40.0;
    }
}

void main() {
    Professor prof = new Professor(20);
    Coordenador coord = new Coordenador();
    prof.CalculaSalario();
    coord.CalculaSalario();
    print("Salario do professor: ${prof.salario}");
    print("Salario do coordenador: ${coord.salario}");
}
```

Console

Salario do professor: 2400

Salario do coordenador: 10000

Para representar herança  
usamos o **extends**

# Dart Intermediário: Coleções

- **Coleções:** são implementações de estruturas de dados, que é utilizado para armazenar itens
- Em dart temos as seguintes coleções:
- Listas: **List**
- Mapas: **Maps**

# Dart Intermediário: Coleções - Listas

- Uma **lista** é uma coleção de valores organizados e com uma ordem, semelhante a um array. Para cada valor há um índice, começando em zero
- Com o objeto **List** é possível usar diversos métodos e propriedades que facilitam a interação e manipulação dos dados

# Dart Intermediário: Coleções - Listas

- Alguns dos métodos:
- `add();` //adicionar valores na lista
- `foreach();` //percorrer a lista toda
- `length` //contar o número de valores armazenados na lista

# Dart Intermediário: Coleções - Listas

```
void main() {  
  List<String> listaFrutas = ["Morango", "Banana", "Laranja"];  
  List<int> listaNumeros = [1, 7, 10];  
  List listaGeral = [1, 2, "Joao", 4, 5];  
  print("\nTodas as frutas: $listaFrutas");  
  print("\nNumeros: $listaNumeros");  
  print("\nNumeros: $listaGeral");  
}
```

▶ RUN

Posso definir ou não o tipo de dados que será armazenado na lista. Para definir indicar o tipo entre <>

Console

Todas as frutas: [Morango, Banana, Laranja]

Numeros: [1, 7, 10]

Numeros: [1, 2, Joao, 4, 5]

# Dart Intermediário: Coleções - Listas

```
void main() {  
  List<String> listaFrutas = ["Morango", "Banana", "Laranja"];  
  
  listaFrutas.add("Pera");  
  listaFrutas.insert(1, "Abacate");  
  print("Frutas: $listaFrutas");  
  listaFrutas.remove("Banana");  
  print("Frutas: $listaFrutas");  
  print("Quantidade de frutas: ${listaFrutas.length}");  
}
```

▶ RUN

Console

```
Frutas: [Morango, Abacate, Banana, Laranja, Pera]  
Frutas: [Morango, Abacate, Laranja, Pera]  
Quantidade de frutas: 4
```

O método `add()`; permite adicionar itens na lista.  
O método `remove()`; remove um item da lista.  
A propriedade `length` mostra a quantidade de itens armazenados na lista

# Dart Intermediário: Coleções - Listas

```
class Usuario {  
  String nome;  
  int idade;  
  Usuario(this.nome, this.idade);  
}  
  
void main() {  
  
  List<Usuario> listaUsuarios = List();  
  listaUsuarios.add( Usuario("Joana", 30) );  
  listaUsuarios.add( Usuario("Pedro", 45) );  
  listaUsuarios.add( Usuario("Maria", 20) );  
  
  for( Usuario usuario in listaUsuarios ){  
    print( "Nome: ${usuario.nome} idade: ${usuario.idade}" );  
  }  
}
```

▶ RUN

Console

```
Nome: Joana idade: 30  
Nome: Pedro idade: 45  
Nome: Maria idade: 20
```

Vamos armazenar na  
lista objetos do tipo  
<Usuario>



# Referências Bibliográficas

- **Flutter:** <https://flutter.dev/>
- **Guia de Flutter:**  
<https://www.devmedia.com.br/guia/flutter/40713>
- **Dart:** <https://dart.dev/>
- **Curso de Dart Gratuito** no Udemy:  
<https://www.youtube.com/watch?v=vl4wfJSvomY>