

Linguagem Dart

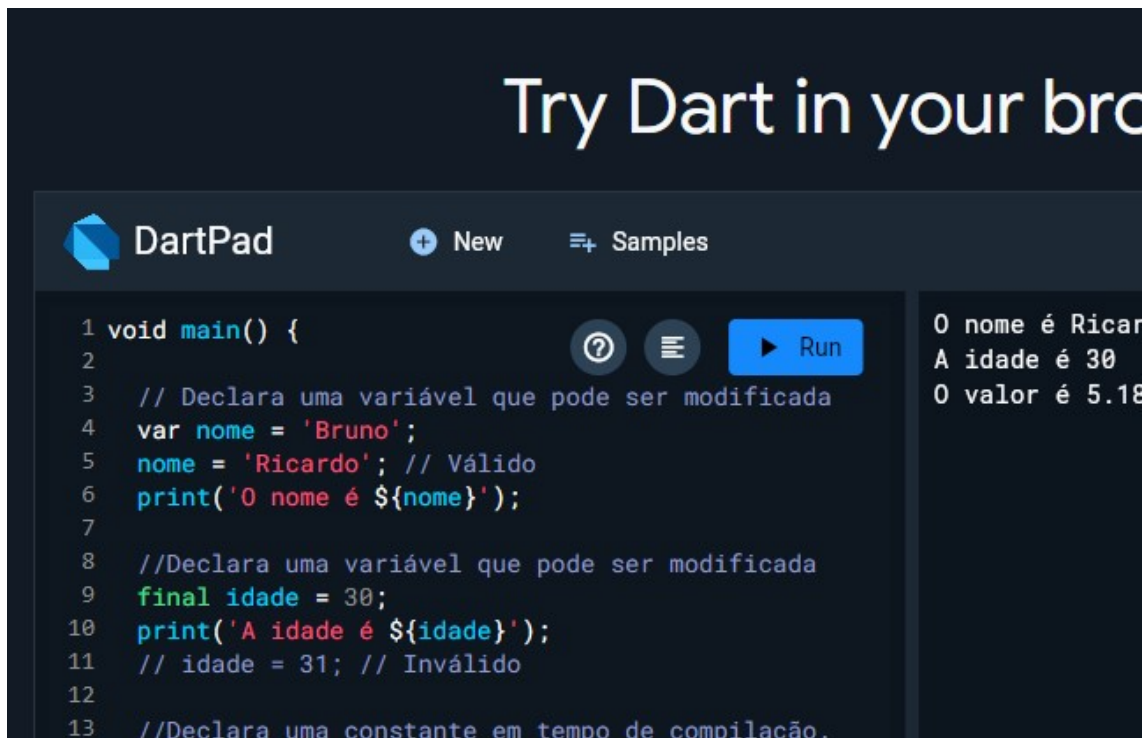
1º) Trabalhando com Variáveis e Tipos de Dados.

Variáveis: Em Dart, você pode declarar variáveis usando as palavras-chave `var`, `final` ou `const`:

var: Declara uma variável que pode ser modificada.

final: Declara uma variável cujo valor não pode ser alterado após a atribuição.

const: Declara uma constante em tempo de compilação.



The screenshot shows the DartPad web interface. At the top, it says "Try Dart in your browser". Below that is the DartPad logo and navigation links for "New" and "Samples". The main area contains a code editor with the following Dart code:

```
1 void main() {  
2  
3   // Declara uma variável que pode ser modificada  
4   var nome = 'Bruno';  
5   nome = 'Ricardo'; // Válido  
6   print('O nome é ${nome}');  
7  
8   //Declara uma variável que pode ser modificada  
9   final idade = 30;  
10  print('A idade é ${idade}');  
11  // idade = 31; // Inválido  
12  
13  //Declara uma constante em tempo de compilação.
```

On the right side of the code editor, there is a "Run" button. Below the code editor, the output of the program is displayed:

```
O nome é Ricardo  
A idade é 30  
O valor é 5.18
```

Tipos de Dados:

Dart é uma linguagem com tipagem estática, mas também suporta inferência de tipos.

Os principais tipos de dados são:

Números: Inteiros (`int`) e de ponto flutuante (`double`).

Strings: Cadeias de caracteres.


Booleanos: Representam valores verdadeiros ou falsos.

Listas: Estruturas de dados que armazenam múltiplos valores.

Conjuntos: Coleções de valores únicos.

Mapas: Estruturas que associam chaves a valores.

Try Dart in your browser

 **DartPad** + New ≡ Samples

```
1 void main() {  
2  
3 //Strings: Cadeias de caracteres.  
4 var nome = 'Ricardo'; // String  
5  
6 //Booleanos: Representam valores verdadeiros ou falsos.  
7 bool isDart = true;  
8  
9 //Números: Inteiros (int) e de ponto flutuante (double).  
10 final idade = 32; // int  
11 const valor = 5.18; // double  
12  
13 //Listas: Estruturas de dados que armazenam múltiplos valores.  
14 List<String> carros = ['Fusca', 'Nivus', 'Opala'];  
15  
16 //Conjuntos: Coleções de valores únicos.  
17 Set<int> numeros = {1, 2, 3};  
18  
19 //Mapas: Estruturas que associam chaves a valores.  
20 Map<String, int> idadePorNome = {'Ricardo': 32, 'João': 38};  
21  
22 print('Linguagem Dart: $isDart');  
23 print('Nome: $nome');
```

?

≡

▶ Run

Linguagem Dart: true
Nome: Ricardo
Idade: 32
Valor: 5.18
Carros: [Fusca, Nivus,
Números: {1, 2, 3}
Idade por Nome: {Ricar

Estrutura de decisão:


Estruturas de decisão permitem que você execute diferentes blocos de código com base em condições. As principais estruturas de decisão são if, else if, else, e switch.

if: Executa um bloco se a condição for verdadeira.



else if e else: Permitem verificar múltiplas condições.

switch: Alternativa para verificar uma variável contra várias opções.



Try Dart in your browser

 **DartPad** + New ≡ Samples



```
1 void main() {
2
3 //if: executa um bloco de código se a condição for verdadeira.
4 int idade = 32;
5
6 if (idade >= 32) {
7   print('Você é maior de idade.');
```

  ▶ Run



```
8 }
9
10 //else if e else: encadear múltiplas condições
11 int nota = 75;
12
13 if (nota >= 90) {
14   print('Aprovado acima da média');
```

  ▶ Run



```
15 } else if (nota >= 75) {
16   print('Aprovado');
```

  ▶ Run



```
17 } else {
18   print('Reprovado');
```

  ▶ Run



```
19 }
20
21 //switch: várias condições verificadas para uma única variável
22 String diaDaSemana = 'quarta';
23
24 switch (diaDaSemana) {
25   case 'segunda':
26     print('Hoje é segunda-feira.');
```

  ▶ Run



```
27     break;
28   case 'terça':
29     print('Hoje é terça-feira.');
```

  ▶ Run

```
30     break;
31   case 'quarta':
32     print('Hoje é quarta-feira.');
```

  ▶ Run

```
33     break;
34   case 'quinta':
```

  ▶ Run


Você é maior de idade.
Aprovado
Hoje é quarta-feira.

2º) Trabalhando com Arrays e Listas.

Listas: são flexíveis e podem armazenar qualquer tipo de dado. Você pode **adicionar**, **acessar**, **remover** e **iterar** sobre elementos facilmente. Dart fornece vários métodos úteis para manipular listas.

Criando Listas: listas vazias e listas com valores iniciais.

Try Dart in your browser

 DartPad + New ≡ Samples

```
1 void main() {  
2  
3 //Listas: Criando listas vazias e com valores  
4 List<int> numerosVazios = [];  
5  
6 // Lista com valores iniciais  
7 List<String> marcas = ['Fiat', 'Jeep', 'Volkswagen'];  
8
```

?


≡

▶ Run

[Fiat, Jeep, Volkswagen]

Adicionando Elementos: método add ou addAll

Try Dart in your browser

 DartPad + New ≡ Samples

```
1 void main() {  
2  
3 //Adicionando Elementos: único e múltiplos elementos  
4 List<String> marcas = ['Fiat', 'Jeep', 'Volkswagen'];  
5  
6 marcas.add('Volvo');  
7 marcas.addAll(['Citroen', 'Peugeot']);  
8
```

?


≡

▶ Run

[Fiat, Jeep, Volkswagen, Volvo,
Citroen, Peugeot]

Removendo Elementos: método remove ou removeAt

Try Dart in your browser

 DartPad + New ≡ Samples

```
1 void main() {  
2  
3 List<String> marcas = ['Fiat', 'Jeep', 'Volkswagen'];  
4  
5 print(marcas); //Lista antes da remoção.  
6  
7 marcas.remove('Volkswagen');  
8 marcas.removeAt(0);  
9
```

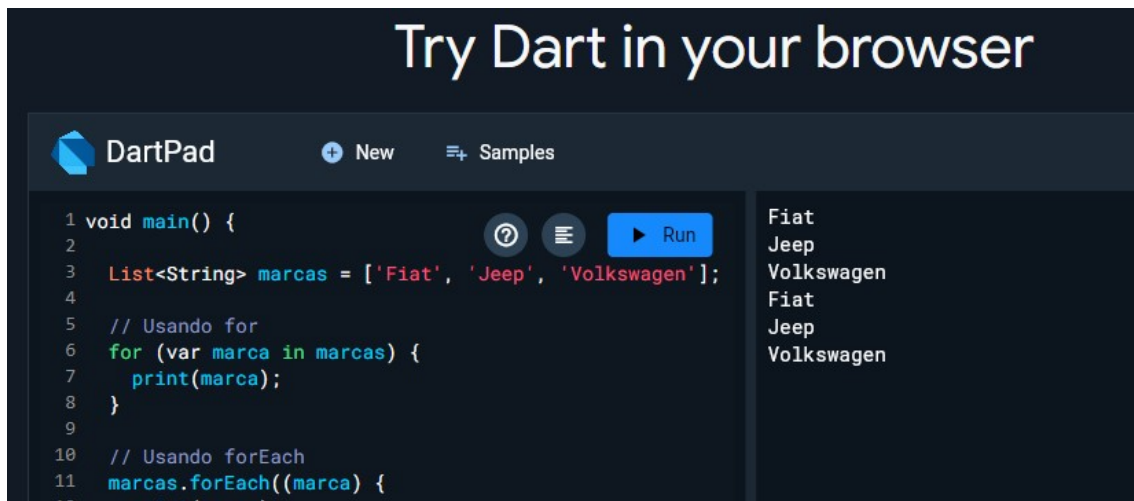
?

≡

▶ Run

[Fiat, Jeep, Volkswagen]
[Jeep]

Iterando sobre Listas: loop for, forEach ou map

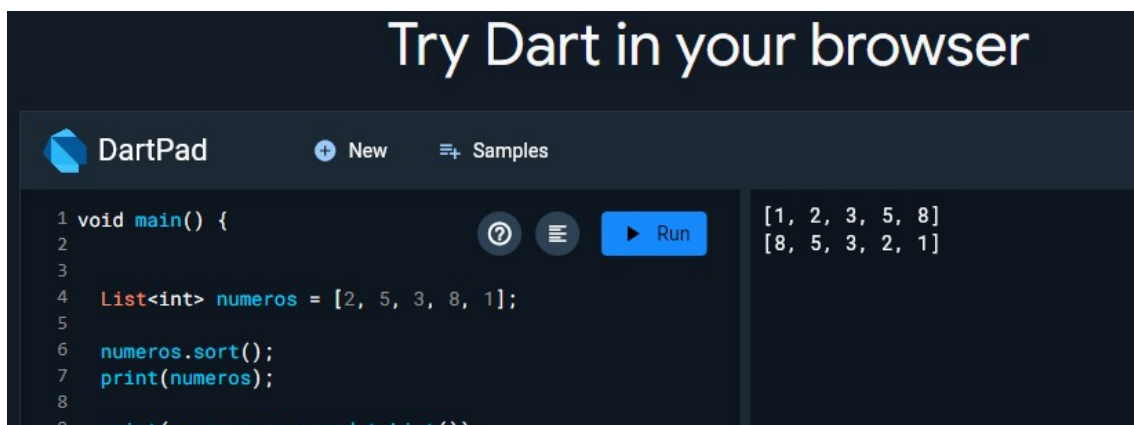


Métodos Úteis:

length: Retorna o número de elementos.

sort(): Ordena os elementos.

reversed: Retorna uma nova lista com os elementos na ordem inversa.




3º) Trabalhando com Opcionais / Null Safety

Null Safety foi introduzido para ajudar a prevenir erros relacionados a valores nulos, tornando o código mais seguro e robusto.

Tipos não nulos e nulos: variáveis em Dart não são nulas, para permitir que uma variável possa ter um valor nulo, você deve usar o operador `?`.

Try Dart in your browser

 DartPad + New ≡ Samples


```
1 void main() {  
2  
3   int idade = 38; // Não pode ser nulo  
4   int? idadeOpcional; // Pode ser nulo  
5  
6   idadeOpcional = null;  
7  
8   print(idade);  
9   print(idadeOpcional);  
}
```

? ≡ ▶ Run

38
null

Usando o Operador de Acesso Condicional: Saída: null (não gera erro).

Try Dart in your browser

 DartPad + New ≡ Samples


```
1 void main() {  
2  
3  
4   String? nome;  
5  
6   // Usando o operador de acesso condicional  
7   print(nome?.length);  
}
```

? ≡ ▶ Run

null

Operador de Coalescência Nula: operador ?? pode ser usado para fornecer um valor padrão caso a variável seja nula.

Try Dart in your browser

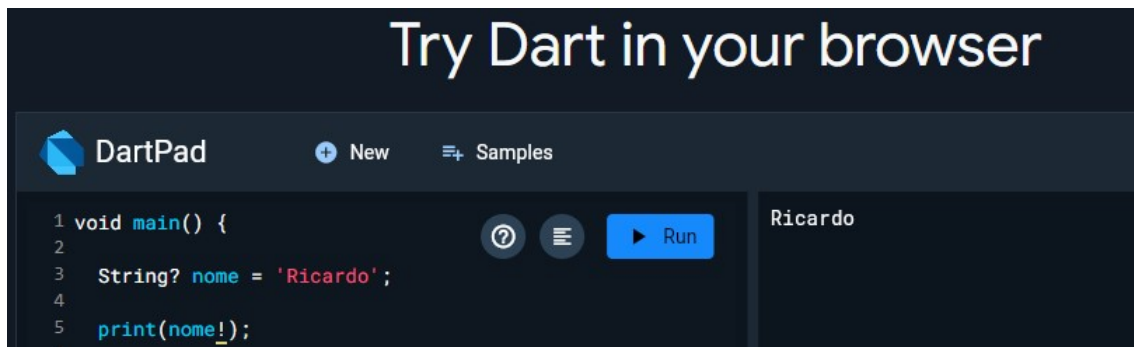
 DartPad + New ≡ Samples

```
1 void main() {  
2  
3  
4   String? nome;  
5  
6   // Usando o operador de acesso condicional  
7   String nomeCompleto = nome ?? 'Desconhecido';  
8   print(nomeCompleto);  
}
```

? ≡ ▶ Run

Desconhecido

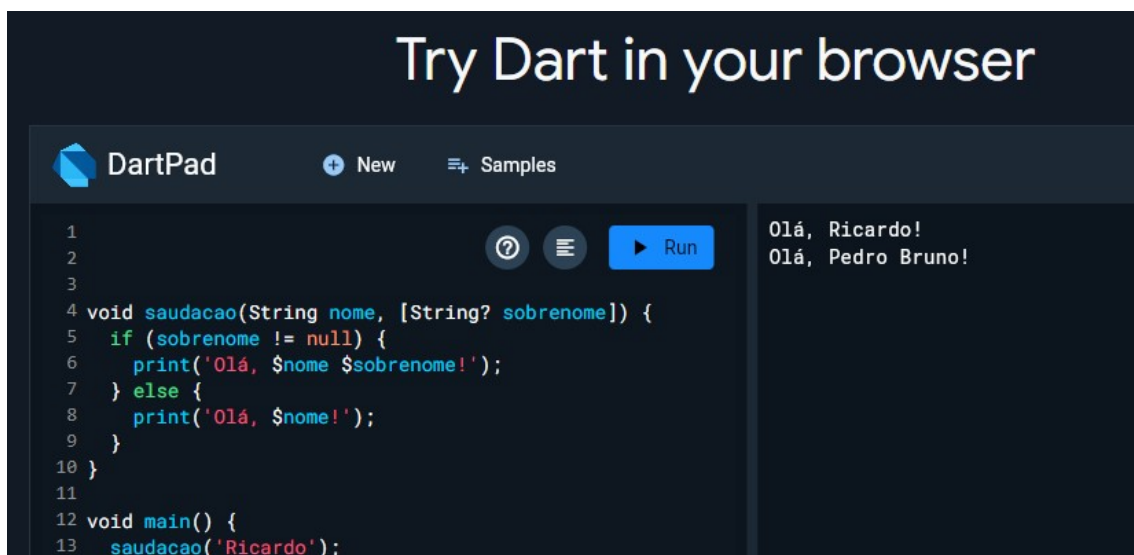
Operador de Aferição Nula: Aferindo que nome não é nulo

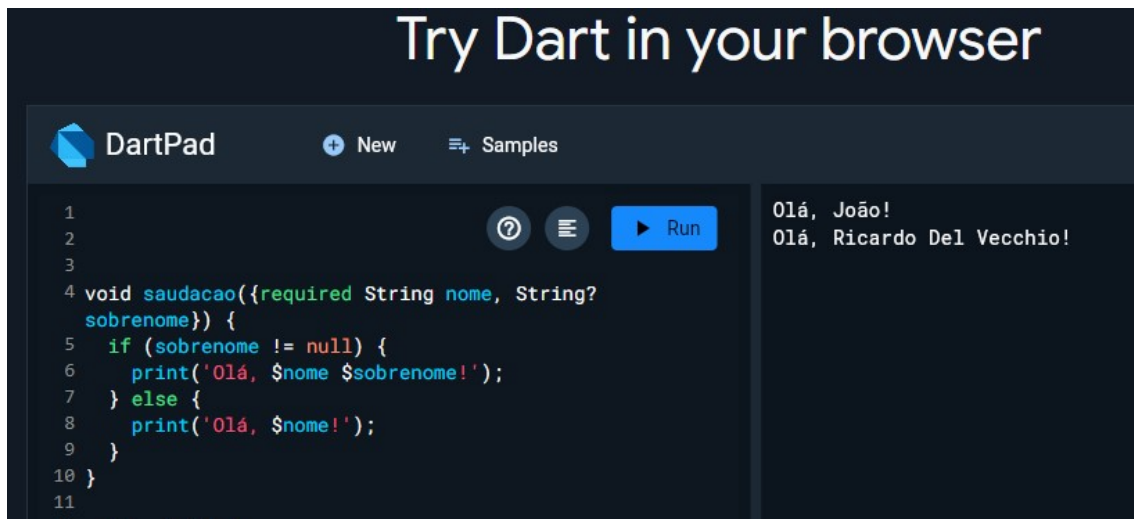


4º) Trabalhando com Funções.

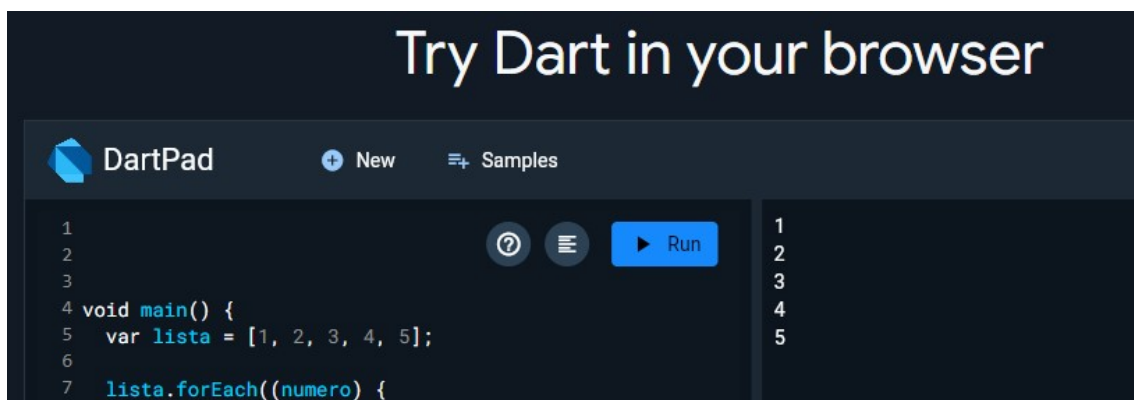
Blocos de código que realizam uma tarefa específica e podem ser reutilizados em diferentes partes do programa. Elas podem aceitar parâmetros, retornar valores e até mesmo ser passadas como argumentos para outras funções.

Funções com Parâmetros Opcionais e Opcionais Nomeados: Definindo parâmetros de função como opcionais, tanto com [] para parâmetros posicionais quanto com {} para parâmetros nomeados.

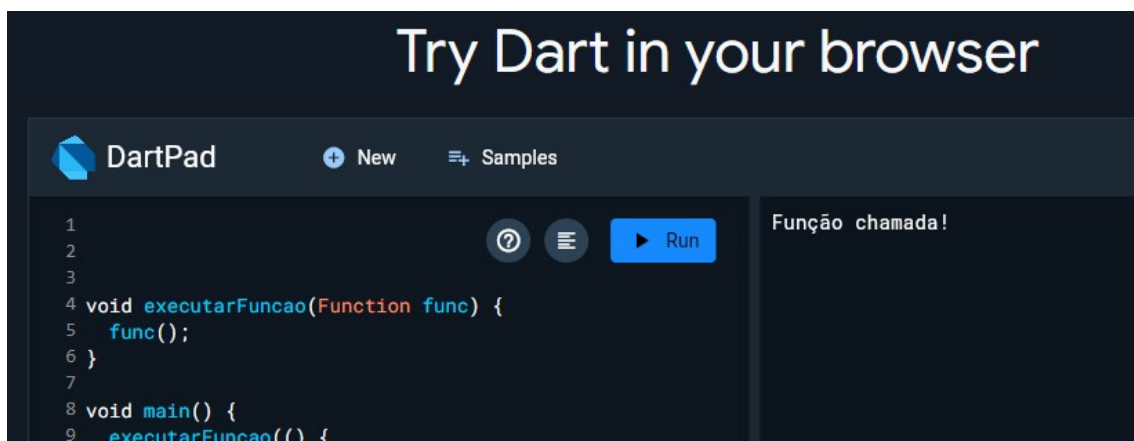




Funções Anônimas (Lambda): f funções sem nome, freqüentemente usadas como callbacks.




Funções como Argumentos: Você pode passar funções como argumentos para outras funções.



Funções de Retorno de Função: Uma função pode retornar outra função.

Try Dart in your browser

 DartPad + New ≡ Samples

```
1
2
3
4 function criaContador() {
5   int contador = 0;
6   return () {
7     contador++;
8     return contador;
9   };
10 }
11
12 void main() {
13   var contador = criaContador();
14 }
```


1
2
3

5º) Trabalhando com POO – Classes.

A Programação Orientada a Objetos (POO) em Dart permite organizar o código em classes e objetos, facilitando a modelagem de sistemas complexos.

Classes: são modelos para criar objetos.

Try Dart in your browser


 DartPad + New ≡ Samples

```
1 class Carro {
2   String modelo;
3   int ano;
4
5   // Construtor
6   Carro(this.modelo, this.ano);
7
8   void mostrarDetalhes() {
9     print('Modelo: $modelo, Ano: $ano');
10  }
11 }
12
```

Modelo: Nivus, Ano: 2023

Construtores: são métodos especiais que são chamados quando um objeto é criado.

Try Dart in your browser


 DartPad + New ≡ Samples

```
1 class Pessoa {
2   String nome;
3   int idade;
4
5   // Construtor nomeado
6   Pessoa(this.nome, this.idade);
7
8   // Construtor padrão
9   Pessoa.anonimo() : nome = 'Desconhecido', idade = 0;
10 }
11
12 void main() {
13   var pessoa1 = Pessoa('Ricardo', 39);
14   var pessoa2 = Pessoa.anonimo();
15 }
```

Ricardo, 39
Desconhecido, 0

Herança: permite que uma classe herde propriedades de outra.

Try Dart in your browser


 DartPad + New ≡ Samples

```
1 class Veiculo {
2   void emitirSom() {
3     print('O veículo faz um som');
4   }
5 }
6
7 class Carro extends Veiculo {
8   @override
9   void emitirSom() {
10    print('Vrum Vrum!');
11  }
12 }
13
```

Vrum Vrum!

Encapsulamento: restringe o acesso a membros da classe, você pode usar modificadores de acesso, como private (usando `_`).

Try Dart in your browser


 DartPad + New ≡ Samples

```
1 class ContaBancaria {
2   String _numeroConta;
3   double _saldo;
4
5   ContaBancaria(this._numeroConta) : _saldo = 0;
6
7   void depositar(double valor) {
8     _saldo += valor;
9   }
10
11  void mostrarSaldo() {
12    print('Saldo: $_saldo');
13  }
14 }
15
16 void main() {
```

Saldo: 350

Métodos estáticos: pertencem à classe e não a instâncias, são úteis para funções que não dependem de um estado específico da instância.

Try Dart in your browser


 DartPad + New ≡ Samples

```
1
2 class MathUtil {
3   static int somar(int a, int b) {
4     return a + b;
5   }
6 }
7
8 void main() {
9   int resultado = MathUtil.somar(25, 35);
```

60

Mixins: permitem reutilizar código em várias classes.

Try Dart in your browser

 DartPad + New ≡ Samples

```
1
2 mixin Corredor {
3   void correr() {
4     print('Correndo!');
5   }
6 }
7
8 class Pessoa with Corredor {}
9
10 void main() {
```

Correndo!