

# DESENVOLVIMENTO DE DISPOSITIVOS MÓVEIS

Diego Possamai

Material Original Profª. Tassiana Kautzmann

# Plug-ins para VS Code

- Dart (fornecedor: Dart Code)
- Flutter (fornecedor: Dart Code)
- Error Lens (fornecedor: Alexander)
- Flutter Color (fornecedor: Nilesh Chavan)
- Flutter Tree (fornecedor: Marcelo Velasquez)

# Flutter

- SDK para desenvolvimento mobile lançado pelo Google;
- Permite a criação de aplicativos para Android e iOS utilizando o mesmo código;
- Os elementos (como botões, menus, estilos, entre outros) são tratados como widgets;
- Linguagem de programação Dart;
- Possui tecnologia **hot reload**, que permite a visualização instantânea das modificações.

# Flutter

- As widgets utilizadas no Flutter, servem tanto para programas antigos quanto para os mais recentes, aumentando a vida útil da plataforma e evita constantes atualizações visuais;
  - Em função disso, os riscos de problemas de compatibilidade com atualizações e diferentes versões de sistemas operacionais, são menores. Se uma empresa lançar um celular mais moderno, os apps desenvolvidos em Flutter continuarão funcionando tranquilamente.

# Introdução ao Dart

Linguagem de programação criada pela Google em 2011;

- Foi criada para substituir o JavaScript, o que não deu muito certo. Porém, com a evolução da linguagem, ela pode ser considerada multiparadigma, embora apresente fortes estruturas típicas de linguagens orientadas a objeto.

O estilo da sua sintaxe é baseada em C, fazendo com que seja similar à Java e C#;

```
main() {  
    print('Hello World!');  
}
```

# OBS.

- **runApp()** é a função que faz com que o Flutter assuma a aplicação de fato para "desenhar na tela" e etc, antes disso, com apenas **void main()** é só o dart que faz alguma coisa;

# Dart – Tipos de dados

- Inteiro:

- Toda e qualquer informação numérica que pertença ao conjunto de números inteiros relativos (números positivos, negativos ou o zero).

```
void main() {  
    int numero = 10;  
    print(numero); // 10  
}
```

- Double:

- Números decimais/fracionados;

```
void main() {  
    double numero_decimal = 10.5;  
    print(numero_decimal); // 10.5  
}
```

# Dart – Tipos de dados

- String:

- Caracteres de texto;

```
void main() {  
    String nome = "Tassiana";  
    print(nome);  
}
```

- Booleano:

- Valores lógicos que podem ser verdadeiro ou falso;

```
void main() {  
    bool vivo = true;  
    print(vivo); // true  
}
```



# Dart – Tipos de dados

- Dynamic:
  - Tipo especial que permite que uma variável possa armazenar qualquer tipo de dado e alterá-lo em tempo de execução;

```
void main() {  
    dynamic dinamico = "Desenvolvimento";  
    print(dinamico);  
    dinamico = 10;  
    print(dinamico);  
}
```

# Dart – Tipos de dados

- List
  - Representa coleção de objetos;
  - É sinônimo de array em outras linguagens.

```
void main() {  
    List frutas = ['maçã', 'pera', 'banana'];  
    List numeros = [1, 2, 3, 4, 5];  
  
    List profissao = new List();  
    profissao.add('programador');  
    profissao.add('medico');  
  
    //lista de tamanho fixo  
    List comidas = new List(3);  
    comidas[0] = "arroz";  
    comidas[1] = "feijão";  
    comidas[2] = "macarrão";  
  
    print("Lista de frutas: $frutas");  
    print("Lista de numeros: $numeros");  
    print("Lista fixa de comida: $profissao");  
    print("Lista fixa de comida: $comidas");  
}
```

# Dart – Tipos de dados

- Map

```
void main() {  
  
    Map nomesEIdades = {  
        'Pedro' : 75,  
        'Carla' : 71,  
        'Alice' : 29  
    };  
    nomesEIdades['Pedro'] = 29;//alterando a idade de pedro  
  
    print("Maps de nomes e idades: $nomesEIdades");  
    print(nomesEIdades['Alice']);  
}
```

# Dart – Lista tipada

- Sintaxe: `List <tipo> nome_lista = ["dados"];`

```
void main() {  
  
    List<String> frutas = [  
        "Limão",  
        "Uva",  
        "Maçã",  
        "Mamão"  
    ];  
  
    print(frutas);  
}
```

# Dart

- Comentários:
  - `//` comentário na mesma linha;
  - `/*` Comentário em  
Várias linhas `*/`

# Dart – Palavras reservadas

<b>abstract</b>	<b>continue</b>	<b>external</b>	<b>implements</b>	<b>operator</b>	<b>this</b>
<b>as</b>	<b>covariant</b>	<b>factory</b>	<b>import</b>	<b>part</b>	<b>throw</b>
<b>assert</b>	<b>default</b>	<b>false</b>	<b>in</b>	<b>rethrow</b>	<b>true</b>
<b>async</b>	<b>deferred</b>	<b>final</b>	<b>interface</b>	<b>return</b>	<b>try</b>
<b>await</b>	<b>do</b>	<b>finally</b>	<b>is</b>	<b>set</b>	<b>typedef</b>
<b>break</b>	<b>dynamic</b>	<b>for</b>	<b>library</b>	<b>show</b>	<b>var</b>
<b>case</b>	<b>else</b>	<b>Function</b>	<b>mixin</b>	<b>static</b>	<b>void</b>
<b>catch</b>	<b>enum</b>	<b>get</b>	<b>new</b>	<b>super</b>	<b>while</b>
<b>class</b>	<b>export</b>	<b>hide</b>	<b>null</b>	<b>switch</b>	<b>with</b>
<b>const</b>	<b>extends</b>	<b>if</b>	<b>on</b>	<b>sync</b>	<b>yield</b>

# Dart – Identificadores

- Nomes dos elementos no programa, como variáveis, funções...
  - Não pode começar com um dígito, mas pode conter dígito;
  - Não pode incluir símbolos especiais, com exceção do “\_” e do “\$”;
  - Não pode conter palavra reservada;
  - Deve ser única;
  - É case-sensitive: “Cadeira” != “cadeira”;
  - Não pode conter espaço;

# Dart – Identificadores

- Diferente do Java, Dart não utiliza palavras-chave public, protected e private;
- Se um identificador começar com “\_”, é privado para sua biblioteca

- `Int _private;`



# Dart – Operadores Aritméticos

Operador	Significado	Exemplo
+	Soma	$3 + 2 = 5$
-	Subtração	$3 - 2 = 1$
*	Multiplicação	$3 * 2 = 6$
/	Divisão	$3 / 2 = 1.5$
~/	Divisão que retorna inteiro	$3 ~/ 2 = 1$
%	Resto da divisão	$3 \% 2 = 1$
++	incrementar em 1	<code>i = 3; i++; // 4</code>
--	decrementar em 1	<code>i = 3; i--; // 2</code>

# Dart – Operadores de igualdade e relacionais

Operador	Significado	Exemplo
>	Maior que	<code>1 &gt; 3 // false</code>
<	Menor que	<code>1 &lt; 3 // true</code>
>=	Maior que ou igual	<code>1 &gt;= 3 // false</code>
<=	Menor que o ou igual	<code>1 &lt;= 3 // true</code>
==	Igualdade	<code>1 == 3 // false</code>
!=	Diferente	<code>1 != 3 // true</code>

# Dart – Operadores de Atribuição

Operador	Significado	Exemplo
=	Atribuição	<code>int a = 3;</code>
??=	Atribuição se a variável for null	<code>int a;a??=3; // a = 3;</code>
+=	Soma e atribui	<code>int a = 3; a+=1; // 4</code>
-=	Subtrai e atribui	<code>int a = 3; a-=1; // 2</code>
*=	Multiplica e atribui	<code>int a = 3; a*=1; // 3</code>
/=	Divide e atribui	<code>int a = 3; a/=1; // 3</code>

# Dart – Operadores lógicos e verificação de tipo

Operador	Significado	Exemplo
<code>&amp;&amp;</code>	AND - Retorna true se as duas expressões forem true	<code>true &amp;&amp; true = true</code>
<code>  </code>	OR - Retorna true se pelo menos uma expressão for true	<code>true    false = true</code>
<code>!</code>	NOT - nega o resultado da expressão	<code>!false = true</code>
<code>is</code>	True se o objeto tem o tipo especificado	<code>int a; a is int // true</code>
<code>is!</code>	False se o objeto tem o tipo especificado	<code>int a; a is! int // false</code>

# Dart

- [Dartpad](#)
- Variáveis e prints
- **código de exemplo 1:**

```
void main() {  
    String nome = 'NomeAqui';  
    int a = 1;  
    int b = 3;  
    int c = a + b;  
    print('Soma: $c');  
    print('Nome: $nome');  
}
```

# Funções

- Declaração

```
void nomeDaFuncao(argumento1, argumento2, ...) {  
    // código a ser executado pela função  
}
```

- Retorno de valor

```
tipoDeValor nomeDaFuncao(argumento1, argumento2,  
...) {  
    // código a ser executado pela função  
    return valor;  
}
```

# Funções

- Parâmetros opcionais

```
void nomeDaFuncao(argumento1, [argumento2]) {  
    // código a ser executado pela função  
}
```

# Funções - Ex

```
void main() {
    int a = 1;
    int b = 3;
    int c = soma(a, b);
    print("Soma: $c");
}

int soma(int a, int b){
    return a + b;
}
```

- O retorno pode ser omitido, podemos utilizar o código desta forma:

```
void main() {
    int a = 1;
    int b = 3;
    int c = soma(a, b);
    print("Soma: $c");
}

soma(a, b){
    return a + b;
}
```



# Dart

- Condicionais: IF / ELSE

```
void main() {  
    double media = 4.9;  
  
    // IF (condição verdadeira) / ELSE  
    if (media < 6.0) {  
        print("Reprovado!");  
    } else {  
        print("Aprovado!");  
    }  
  
    /* Podemos também utilizar IF TERNÁRIO  
    * CONDIÇÃO ? RETORNO VERDADEIRO : RETORNO FALSO  
    */  
    print(media < 6.0 ? "Reprovado!" : "Aprovado");  
}
```



## IF TERNÁRIO

É uma forma compacta  
de realizar um  
**if-else**.

# Dart

- Condicionais: SWITCH

```
void main(){  
    String linguagem;  
    linguagem = "Dart";  
    switch(linguagem){  
        case "Dart":  
            print("É Dart!");  
            break;  
        case "Java":  
            print("É Java!");  
            break;  
        case "C#":  
            print("É C#!");  
            break;  
        default:  
            print("Não sabe no que programa");  
    }  
}
```

# Dart

- Repetições: FOR

```
void main() {  
    // Repetição de 0 a 5  
    // FOR (INICIO; CONDIÇÃO; INCREMENTO)  
    for(int i = 0; i < 5; i++){  
        print(i);  
    }  
}
```

- WHILE

```
void main() {  
    // Repetição de 0 a 5  
    // INICIO; WHILE (CONDICAO){ INCREMENTO; }  
    // Teste condicional no início  
    int j = 0;           // Início  
    while(j < 5) {       // Condição  
        print(j);  
        j++;             // Incremento  
    }  
}
```

# Dart

- Repetições: DO WHILE

```
void main() {  
    // Repetição de 0 a 5  
    // INICIO; DO { INCREMENTO; } WHILE(CONDICA0);  
    // Teste condicional no final  
    int k = 0;          // Início  
    do {  
        print(k);  
        k++;            // Incremento  
    } while(k < 5);     // Condição  
}
```

# NULL SAFETY



# Null Safety

- Nas versões mais atuais do dart, já está sendo aplicado o conceito **null safety**, o que significa que as variáveis não podem ser nulas a menos que seja especificado que ela seja nula;
- <https://pub.dev>
- O objetivo é evitar erros do tipo null.

# Null Safety

- Para podermos utilizar uma variável sem precisar inicializá-la, podemos utilizar o “?” após definirmos o seu tipo. Ex:
  - `String? Nome;`
  - Outra forma é utilizar o “late”, indicando que a variável não é nula, mas será inicializada mais tarde. Ex:
    - `Late String nome;`
- Saiba mais:
  - <https://www.linkedin.com/pulse/dart-null-safety-tudo-que-você-precisa-saber-gomes-vieira/?originalSubdomain=pt>

# Orientação a Objetos com Dart

- Como é uma classe em java:

```
public class Contato {  
  
    String nome;  
    String email;  
    String endereco;  
  
    public Contato(String nome, String email, String endereco) {  
        super();  
        this.nome = nome;  
        this.email = email;  
        this.endereco = endereco;  
    }  
}
```



# Orientação a Objetos com Dart

## Como é uma classe em dart:

- Forma parecida com java:

```
class Contato{

    String? nome;
    String? email;
    String? endereco;

    Contato(){
        this.nome = nome;
        this.email = email;
        this.endereco = endereco;
    }

}
```

- Forma simplificada:

```
class Contato{

    String? nome;
    String? email;
    String? endereco;

    Contato({this.nome, this.email, this.endereco});

}
```

# Orientação a Objetos com Dart

- Construtor em dart:
  - Construtor normal:

```
main(){  
  
    Contato c = Contato('Tassi', 'tassik@gmail.com', 'rua tal');  
  
    print('O nome é ${c.nome}');  
  
}  
  
class Contato{  
  
    String? nome;  
    String? email;  
    String? endereco;  
  
    Contato(this.nome, this.email, this.endereco);  
}
```

# Orientação a Objetos com Dart

- Construtor em dart:
  - Construtor com parâmetros nomeados: quando instanciar o objeto, é obrigatório declarar, utilizando {}:

```
main(){
  Contato c = Contato(
    nome: 'Tassiana',
    email: 'tassik@gmail.com',
    endereco: 'rua tal'
  );

  print('O nome é ${c.nome}');
}

class Contato{

  String? nome;
  String? email;
  String? endereco;

  Contato({this.nome, this.email, this.endereco});
}
```

# Orientação a Objetos com Dart

- Forma simplificada com “required”:

```
class Contato{  
    String? nome;  
    String? email;  
    String? endereco;  
  
    Contato({required this.nome, this.email, this.endereco});  
}
```

# Orientação a Objetos com Dart

- No [dartpad](#):

```
main() {  
  Contato c = Contato(  
    nome: 'Tassiana',  
    email: 'tassik@gmail.com',  
    endereco: 'rua tal'  
  );  
  
  Contato c1 = Contato();  
  c1.nome = 'Ana';  
  
  print('O nome é ${c.nome}');  
  print('O nome é ${c1.nome}');  
}
```

```
class Contato{  
  
  String? nome;  
  String? email;  
  String? endereco;  
  
  Contato({this.nome, this.email,  
    this.endereco});  
  
  //Contato({required this.nome, this.email,  
    this.endereco});  
}
```

# Orientação a Objetos com Dart

- Visibilidade:
  - `_private`.
  - No get o retorno é implícito.

```
class Contato{  
  
    late String _nome;  
  
    //Contato({this.nome}); não pode ser usado parâmetro nomeado  
    //com visibilidade privada  
  
    Contato(this._nome);  
  
    get nome => this._nome;  
    set nome(nome) => this._nome = nome;  
}
```