

# **DCC917A – TÓPICOS ESPECIAIS III: DESENVOLVIMENTO DE APLICATIVOS MÓVEIS**

**AULA 03**

**Carlos Bruno Oliveira Lopes**

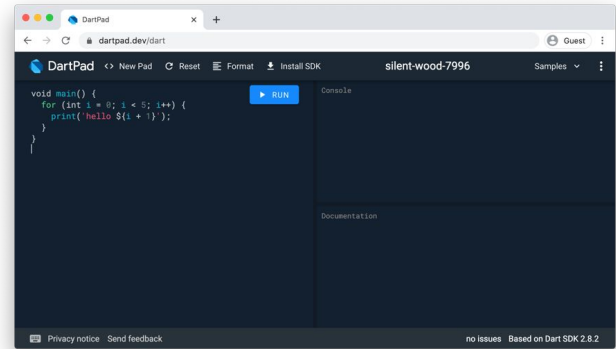
*Engenheiro de Computação*

*Mestre em Ciência da Computação*

# Desenvolvimento de aplicativos móveis

## Programando em Dart

- Para o aprendizado da linguagem Dart usaremos o DartPad
  - DartPad é uma ferramenta que permite “brincar” com linguagem Dart em qualquer navegador web;
  - Ele pode ser acessado por meio do link:
    - <https://dartpad.dartlang.org/>
    - Usaremos essa plataforma para testarmos códigos em Dart



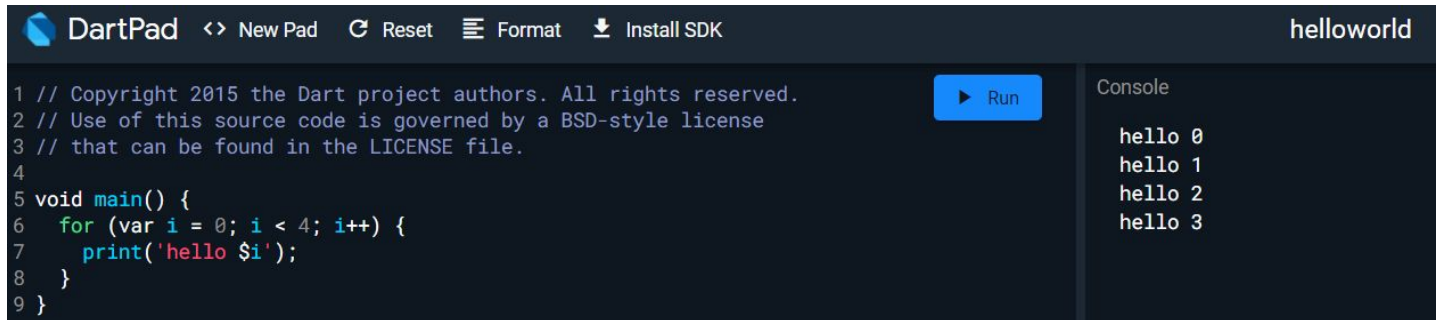
# Desenvolvimento de aplicativos móveis



The image shows a screenshot of the DartPad web interface. At the top, there is a dark blue header bar with the DartPad logo on the left and several navigation buttons: '<> New Pad', a circular arrow icon for 'Reset', a hamburger menu icon for 'Format', and a download icon for 'Install SDK'. Below the header, the main area is a dark blue editor with light blue text. It contains a Dart program with 9 lines of code. The code starts with three comment lines, followed by an empty line, then a 'void main()' function. Inside the function, there is a 'for' loop that iterates from 0 to 3, printing 'hello \$i' for each value of 'i'. The code is color-coded: keywords like 'void', 'main', 'for', and 'print' are in blue, variables like 'i' are in green, and string literals like 'hello \$i' are in red.

```
1 // Copyright 2015 the Dart project authors. All rights reserved.  
2 // Use of this source code is governed by a BSD-style license  
3 // that can be found in the LICENSE file.  
4  
5 void main() {  
6   for (var i = 0; i < 4; i++) {  
7     print('hello $i');  
8   }  
9 }
```

# Desenvolvimento de aplicativos móveis



The image shows a screenshot of the DartPad web editor. The top bar includes the DartPad logo and navigation links: '<> New Pad', 'Reset', 'Format', and 'Install SDK'. The file name 'helloworld' is displayed on the right. The main editor area contains the following Dart code:

```
1 // Copyright 2015 the Dart project authors. All rights reserved.  
2 // Use of this source code is governed by a BSD-style license  
3 // that can be found in the LICENSE file.  
4  
5 void main() {  
6   for (var i = 0; i < 4; i++) {  
7     print('hello $i');  
8   }  
9 }
```

A blue 'Run' button is located to the right of the code. The console on the right side shows the output of the program:

```
hello 0  
hello 1  
hello 2  
hello 3
```

# Linguagem de programação Dart

## Comentário

– O Dart tem suporte a três tipos de comentários:

1. Comentários de linhas por meio do `//` :

```
// Define a idade do usuário  
int age = 25; // A idade é 25
```

2. Comentários multilinhas por meio do `/*` e `*/` :

```
/*  
Essa função calcula o saldo da conta  
usando o método de Miller-Hawthorne  
de cálculo de valor  
*/
```

# Linguagem de programação Dart

## Comentário

– O Dart tem suporte a três tipos de comentários:

3. Comentários de documentação. Pode ser em linha `///` ou em multiplas linhas `/**` e `*/`:

```
/// Esse é um comentário de documentação
/**
  Esse também
  é um
  comentário de documentação
*/
```

# Linguagem de programação Dart

## Comentário

– O Dart tem suporte a três tipos de comentários:

### 3. Comentários de documentação.

- Exceção no uso `///`, podemos usar o colchetes `[ ]` nesse comentário para gerar um referência no código, ou seja, podemos vincular um trecho do código ao comentário.

```
class Pet {  
  num legs;  
  /// Alimenta o animal [Treats]  
  feed(Treats treat) {  
    // Alimenta a criatura!  
  }  
}
```

Nesse exemplo, quando a documentação for gerada (o que você pode fazer com a ferramenta `dartdoc` do Dart SDK), o texto `[Treats]` passará a ser um link para a documentação da classe `Treats`

# Linguagem de programação Dart

## Variáveis

- As variáveis no Dart, armazenam um valor ou uma referência;
- Tudo em Dart é objeto;
  - números, strings e mesmo null, são todos objetos, ou seja, são instâncias de classes.
    - Todos estendem da classe comum Object.
- Identificadores podem começar com uma letra ou underscore;
  - No entanto, quando começam com underscore eles tem um significado especial: é privado da biblioteca (ou classe) em que se encontra.
    - O Dart não possui palavras-chaves de visibilidade, mas, o underscore possui o mesmo efeito que um private tem em Java.



# Linguagem de programação Dart

## Declaração e inicialização de variáveis

– A declaração no Dart pode ser feito de duas formas:

`var x;` (Dart inferirá o tipo a partir do valor atribuído)

`<algun tipo específico> x;` (Declaração explícita)

Ex.:

`var x;` (Nessa declaração a **x** terá valor igual a **null**)

`var x = "Mel Brooks";`

`String x = "Mel Brooks";` (**x** é uma referência a uma String)

# Linguagem de programação Dart

## Declaração e inicialização de variáveis

- Há uma terceira opção:

```
dynamic x = "Mel Brooks";
```

- A anotação de tipo **dynamic** informa ao Dart que o que x referencia pode mudar com o tempo.

Ex.:

```
dynamic x = "Mel Brooks";
```

```
x = 42;
```

(Dart não reclamará de x agora estar apontando para um valor numérico em vez de uma string.)

- Há também uma quarta e última opção de declaração:

```
object x = "Mel Brooks";
```

# Linguagem de programação Dart

## Constantes e valores final

- As palavras-chaves **const** e **final** definem uma variável como sendo uma constante, um valor final imutável, respectivamente.

```
const x = "Mel Books";  
const String x = "Mel Brooks";  
final x = "Mel Brooks";
```

### – Diferença entre **final** e **const**

- **final** significa que você só pode definir a variável uma vez, mas pode fazê-lo no tempo de execução;
- **const** indica que ela só pode ser definida uma vez, mas seu valor já deve ser conhecido no tempo de compilação.

# Linguagem de programação Dart

## Constantes e valores final

- A palavra-chave **const** permite ao programador aplicá-la tanto a valores quanto a variáveis.

Ex.:

```
List lst = const [ 1, 2, 3];  
print(lst);  
lst = [ 4, 5, 6 ];  
print(lst);  
lst[0] = 999;  
print(lst);
```

**Obs.:** Se a linha `lst[0] = 999` fosse movida, haveria uma exceção porque se esta tentando alterar uma lista que foi referenciada como `const`

# Linguagem de programação Dart

## Tipos de dados

- Dart é uma linguagem fortemente tipada;
  - Ele permite a não explicitação dos tipos. Eles são opcionais;
  - Ele consegue executar a inferência de tipos quando as anotações não estão presentes;

# Linguagem de programação Dart

## Tipos de dados

### String

- O tipo String é uma sequência de caracteres de código UTF-16;
- Elas podem ser usadas com aspas simples ou duplas;
- Elas podem incluir expressões externas (de outras variáveis ou objetos) por meio do uso da sintaxe **`${expressão}`**.
  - É possível omitir as chaves quando a expressão referenciar um identificador

Ex.:

```
String s1 = "Rickety Rocket";
String s2 = "${s1} blast off!";
String s3 = '$s1 blast off!';
print (s2);
print (s3);
```

# Linguagem de programação Dart

## Tipos de dados

### String

- A concatenação de Strings pode ser realizada por meio do operador **+**;  
Ex.: `var a = 'b' + 'c';` (resulta em 'bc')
- Ou forma de concatenar Strings é por meio de strings literais adjacentes;  
Ex.: `var b = 'b' ` ` c';` (resulta em 'b c')

# Linguagem de programação Dart

## Tipos de dados

### Valores numéricos

- Tipo **int** são valores numéricos inteiros
  - O intervalo de valores é  $-2^{63}$  a  $2^{63} - 1$  na VM Dart;
- Tipo **double** são valores numéricos de ponto flutuante de precisão dupla
  - Utiliza o padrão IEEE 754
- Tanto **int** quanto **double** são subclasses de **num**
  - Isso permite definir um variável apenas usando o num tanto para inteiro quanto para ponto flutuante que o Dart identificar se é int ou double baseado no valor atribuído a variável.

Ex.:

```
num w = 5;  
num x = 5.5;  
int y = 5;  
double z = 5.5;
```



# Linguagem de programação Dart

## Tipos de dados

### Valores numéricos

- Valores numéricos podem ser transformado em uma string com o uso do método **toString()** das classes **int** e **double** (ou **num**)

Ex.:

```
int i = 5;
double d = 5.5;
String si = i.toString();
String sd = d.toString();
print(i);
print(d);
print(si);
print(sd);
```

# Linguagem de programação Dart

## Tipos de dados

### Valores numéricos

- Uma string pode ser transformada em um número com o método **parse()** das classes **int** e **double** (ou **num**)

Ex.:

```
String si = "5";  
String sd = "5.5";  
int i = int.parse(si);  
double d = double.parse(sd);  
print(si);  
print(sd);  
print(i);  
print(d);
```

# Linguagem de programação Dart

## Tipos de dados

### Valor booleano

- Tipo **bool** representam valores booleanos.
  - Há dois objetos que representam os valores booleanos: **true** e **false** (são constantes do tempo de compilação)

# Linguagem de programação Dart

## Tipos de dados

### Lista

- A classe **List** é semelhante ao array de outras linguagem
    - Sua instância é uma lista de valores
- Ex.:
- ```
List lst = [1, 2, 3];  
var lst1 = [ 1, 2, 3 ];  
Object lst2 = [ 1, 2, 3 ];
```
- Uma lista usa um esquema de indexação baseado em zero
  - O tamanho de uma lista pode ser obtido usando o método length
- Ex.:
- ```
List lista = [0, 1, 2, 3];  
print (lista.length);
```
- Para acessar um elemento na lista basta indicar o índice na lista
- Ex.:
- ```
print (lst[1]);
```

# Linguagem de programação Dart

## Tipos de dados

### Lista

Ex.:

```
List lst = [ 8, 3, 12 ];  
lst.add(4);                (Adicionar um elemento no final da lista)  
lst.sort((a, b) => a.compareTo(b)); (ordena os elementos na lista)  
lst.removeLast();          (Remove o ultimo elemento na lista)  
print(lst.indexOf(4));      (Localiza a posição de um elemento na lista)  
print(lst);
```

# Linguagem de programação Dart

## Tipos de dados

### Set

- Classe **Set** é semelhante a List, mas é uma lista não ordenada
  - Isso significa que não é possível recuperar elementos pelo índice, portanto, precisa-se usar os métodos **contains()** e **containsAll()**

Ex.:

```
Set cookies = Set();
cookies.addAll([ "oatmeal", "chocolate", "rainbow" ]);
cookies.add("oatmeal"); // Se não há dano, não há problema
cookies.remove("chocolate");
print(cookies);
print(cookies.contains("oatmeal"));
print(cookies.containsAll([ "chocolate", "rainbow" ]));
```

- » A chamada a `contains()` retorna `true`, enquanto a chamada a `containsAll()` retorna `false` já que `chocolate` foi removido com `remove()`.

# Linguagem de programação Dart

## Tipos de dados

### Map

- Classe **Map** é um dicionário (ou hash)

Ex.:

```
var actors = {  
    "Ryan Reynolds" : "Deadpool",  
    "Hugh Jackman" : "Wolverine"  
};  
print(actors);  
var actresses = new Map();  
actresses["scarlett johansson"] = "Black Widow";  
actresses["Zoe Saldana"] = "Gamora";  
print (actresses);
```

# Linguagem de programação Dart

## Tipos de dados

### Map

- Classe **Map** é um dicionário (ou hash)

Ex.:

```
var movies = Map<String, int>();  
movies["Iron Man"] = 3;  
movies["Thor"] = 3;  
print(movies);  
print(actors["Ryan Reynolds"]);  
print(actresses["Elizabeth Olsen"]);  
movies.remove("Thor");  
print(movies);  
print(actors.keys);  
print(actresses.values);
```



# Linguagem de programação Dart

## Tipos de dados

### Map

- Classe **Map** é um dicionário (ou hash)

Ex.:

```
Map sequels = { };  
print(sequels.isEmpty);  
sequels["The Winter Soldier"] = 2;  
sequels["Civil War"] = 3;  
sequels.forEach((k, v) {  
    print(k + " sequel #" + v.toString());  
});
```

# Linguagem de programação Dart

## Tipos de dados

### Map

- Classe **Map** é um dicionário (ou hash)

Ex.:

```
Map sequels = { };  
print(sequels.isEmpty);  
sequels["The Winter Soldier"] = 2;  
sequels["Civil War"] = 3;  
sequels.forEach((k, v) {  
    print(k + " sequel #" + v.toString());  
});
```

# Linguagem de programação Dart

## Tipos de dados

### Map

- Classe **Map** é um dicionário (ou hash)

Ex.:

- » O primeiro mapa, `actors`, é criado com o uso de chaves e com dados definidos imediatamente dentro dele.
- » O segundo, `actresses`, usa a palavra-chave `new` para criar uma nova instância de `Map` explicitamente.
  - Os elementos são adicionados a ele com o uso da notação de colchete, em que o valor dentro do colchete é a chave e o valor após o sinal de igualdade é o que é mapeado para essa chave.
- » A terceira defini os tipos para as chaves e valores de um mapa.
- » O quarto mapa, `sequels`, define um tipo `Map` criando uma lista de chaves e valores.

# Linguagem de programação Dart

## Tipos de dados

### Map

- Classe **Map** é um dicionário (ou hash)

Ex.:

- » Os métodos **keys** e **values**, retorna as lista de chaves e a lista de valores, respectivamente.
- » O método **remove()** remove um elemento de um mapa.
- » O método **isEmpty()** informa se o mapa está ou não vazio.
- » O método **forEach()** permite executar uma função arbitrária para cada elemento do mapa.
  - A função fornecida receberá a chave e o valor.

# Linguagem de programação Dart

## Tipos de Dados

### Enumeração

- Usado quando há a necessidade de objeto com uma quantidade fixa de valores constantes sem precisar criar classes e variáveis para isso.

Ex.:

```
enum SciFiShows { Babylon_5, Stargate_SG1, Star_Trek }
main() {
  print(SciFiShows.values);
  print(SciFiShows.Stargate_SG1.index);
  var show = SciFiShows.Babylon_5;
  switch (show) {
    case SciFiShows.Babylon_5: print("B5"); break;
    case SciFiShows.Stargate_SG1: print("SG1"); break;
    case SciFiShows.Star_Trek: print("ST"); break;
  }
}
```

# Linguagem de programação Dart

## Tipos de Dados

### void

- Usado geralmente para em funções para indicar um a mesma não retorna nada
  - No entanto, no Dart podemos **omiti-lo em funções** que não retornam nada.
    - » Nessa casos, o Dart inseri a instrução `return null`.
- O tipo void pode ser usado como parâmetros de tipo genérico, onde eles forem semanticamente tratados como Object.

Ex.:

```
main() {  
  List<void> l = [ 1, 2, 3 ];  
  // Equivalente a List<Object> = [ 1, 2, 3 ];  
  print(l);  
}
```

# Linguagem de programação Dart

## Palavras-chaves

### is

- Permite determinar se uma referência é de um tipo específico.

Ex.:

```
if (shape is Circle) {  
    print(circle.circumference);  
}
```

### as

- Permite converter (se possível) e/ou verificar se referência é de um tipo específico

Ex.:

```
(shape as Circle).circumference = 20;  
▪ O código funcionará se shape for do tipo Circle, e/ou, se shape puder ser convertido no tipo Circle.
```

# Estruturas de controle de fluxo no Dart

## Estrutura de repetição (*looping*)

- As estruturas de repetição são realizadas por meio do **for**, **while** ou **do while**.

### **for**

Ex.:

```
for (var i = 0; i < 10; i++) {  
  print(i);  
}
```



# Estruturas de controle de fluxo no Dart

## Estrutura de repetição (*looping*)

- As estruturas de repetição são realizadas por meio do **for**, **while** ou **do while**.

### **for**

- **for in**, aplicado a classe que são iteráveis

Ex.:

```
List starfleet = [ "1701", "1234", "1017", "2610", "7410" ];  
main() {  
    for (var shipNum in starfleet) {  
        print("NCC-" + shipNum);  
    }  
}
```

# Estruturas de controle de fluxo no Dart

## Estrutura de repetição (*looping*)

- As estruturas de repetição são realizadas por meio do **for**, **while** ou **do while**.

### **for**

- **for in**, aplicado a classe que são iteráveis,
- **foreach**, método (função) de classe semelhante ao **for in**.

Ex.:

```
List starfleet = [ "1701", "1234", "1017", "2610", "7410" ];  
main() {  
    starfleet.forEach((shipNum) => print("NCC-" + shipNum));  
}
```

# Estruturas de controle de fluxo no Dart

## Estrutura de repetição (*looping*)

- As estruturas de repetição são realizadas por meio do **for**, **while** ou **do while**.

### **while**

Ex.:

```
while (!isDone()) {  
    // Faz algo  
}
```

# Estruturas de controle de fluxo no Dart

## Estrutura de repetição (*looping*)

- As estruturas de repetição são realizadas por meio do **for**, **while** ou **do while**.

### **do while**

Ex.:

```
do {  
    showStatus();  
} while (!processDone());
```

# Estruturas de controle de fluxo no Dart

## Estrutura de múltipla seleção

switch

Ex.:

```
switch (someVariable) {  
  case 1:  
    // Faz algo  
    break;  
  case 2:  
    // Faz outra coisa  
    break;  
  default:  
    // Não era 1 ou 2 break;  
}
```

# Estruturas de controle de fluxo no Dart

## Estrutura de seleção ou condicionais

`if`

Ex.:

```
if (mercury == true || venus == true || earth == true || mars
    == true ){
  print ("It's an inner planet");
} else if (jupiter || saturn || uranus || neptune) {
  print ("It's an outar planet");
} else {
  print("Poor Pluto, you are NOT a planet");
}
```

# Linguagem de programação Dart

## Operadores

| Operador | Significado                  |
|----------|------------------------------|
| +        | Adição                       |
| -        | Subtração                    |
| - Expr   | Menos unário prefixado       |
| *        | Multiplicação                |
| /        | Divisão                      |
| ~/       | Divisão inteira              |
| %        | Resto da divisão de inteiros |
| ++var    | Incremento prefixado         |
| var++    | Incremento posfixado         |
| --var    | Decremento prefixado         |
| var--    | Decremento posfixado         |

| Operador | Significado      |
|----------|------------------|
| ==       | Igual a          |
| !=       | Diferente de     |
| >        | Maior que        |
| <        | Menor que        |
| >=       | Maior ou igual a |
| <=       | Menor ou igual a |
| =        | Atribuição       |
| &        | AND lógico       |
|          | OR lógico        |
| &&       | Operador AND     |
|          | Operador OR      |

| Operador  | Significado                                                                            |
|-----------|----------------------------------------------------------------------------------------|
| ^         | XOR lógico                                                                             |
| ~exp      | Complemento unário bitwise                                                             |
| !         | Operador Not                                                                           |
| <<        | Deslocamento para esquerda                                                             |
| >>        | Deslocamento para a direita                                                            |
| a ? b : c | Expressão condicional ternária                                                         |
| a ?? b    | Expressão condicional binária: se a não for nulo, retorna a; caso contrário, retorna b |
| ..        | Notação cascata                                                                        |
| ()        | Aplicação de função                                                                    |
| []        | Acesso a lista                                                                         |
| .         | Acesso a membro                                                                        |

# Linguagem de programação Dart

## Operadores

- Operador `==` faz verificação de valor e não uma verificação de objeto.
  - Quando for necessário verificar se duas variáveis referenciam o mesmo objeto, pode se usar a função global `identical()`.
- Operador `??=` executa uma atribuição se o operando for `null`.
  - `objeto ??= valor` [se o objeto for null, valor será atribuído]
- Operador `?.` Permite o acesso a um membro se o objeto não for null.
  - `objeto?.valor` [Se objeto for não for NULL avalia valor;]
  - `objeto?.metodo()` [Chama um método de um objeto se o objeto Não for NULL;]
  - `objeto?.valor1?.valor2?.metodo()`  
[Se objeto ou valor1 ou valor2 forem NULL a expressão inteira é null caso contrário executa o método;]



# Linguagem de programação Dart

## Operadores

- Operador `..` simplifica a maneira de alterar as propriedades do objeto.

Ex.:

```
Point p = Point();  
p.x = 3;  
p.y = 6;
```



```
Point p = Point();  
..x = 3  
..y = 6;
```

- Operador `tipo?` para criar atributos iniciadas com valor null.

Ex.: `String?`

# Linguagem de programação Dart

## Orientação a objetos no Dart

- O Dart é orientado a objetos, ou seja, classes e objetos.
- Para definir uma classe, basta:

```
class Hero {}
```

- Classe completa:

```
class class_name {  
    <fields>  
    <getters/setters>  
    <constructors>  
    <functions>  
}
```

- **Campos** - é qualquer variável declarada em uma classe. Os campos representam atributos do objeto.
- **Setters e Getters** - Permite que o programa inicialize e recupere os valores dos campos de uma classe.
- **Construtores** - responsáveis por alocar memória para os objetos da classe.
- **Funções** - Funções representam ações que um objeto pode executar, chamados de métodos.

# Linguagem de programação Dart

## Orientação a objetos no Dart

- O Dart é orientado a objetos, ou seja, classes e objetos.
- Para definir uma classe, basta:

```
class Hero {}
```

- Classe completa:

```
class class_name {
```

```
<fields>
```

```
<getters/setters>
```

```
<constructors>
```

```
<functions>
```

```
}
```

**MEMBROS DOS DADOS  
DA CLASSE**

- **Campos** - é qualquer variável declarada em uma classe. Os campos representam atributos do objeto.
- **Setters e Getters** - Permite que o programa inicialize e recupere os valores dos campos de uma classe.
- **Construtores** - responsáveis por alocar memória para os objetos da classe.
- **Funções** - Funções representam ações que um objeto pode executar, chamados de métodos.

# Linguagem de programação Dart

Variáveis de instância (atributos, membros, campos ou propriedades)

- Atributos representam os dados da classe

Ex.:

```
class Hero {  
    String? firstName;  
    String? lastName;  
}
```

# Linguagem de programação Dart

Variáveis de instância (atributos, membros, campos ou propriedades)

- Variáveis de instância podem ser estáticas permitindo que elas possam ser usadas sem a necessidade de instanciar a classe

Ex.:

```
class MyClass {  
    static String greeting = "Hi";  
}  
main() {  
    print(MyClass.greeting);  
}
```

# Linguagem de programação Dart

## Métodos

- Funções que representam ações que objeto pode realizar

Ex.:

```
class Hero {  
  //atributos  
  String? firstName;  
  String? lastName;  
  
  //métodos  
  String sayName() {  
    return "$lastName, $firstName";  
  }  
}
```

# Linguagem de programação Dart

## Métodos

- Podemos criar métodos estáticos com a mesma ideia apresentada para os atributos.

Ex.:

```
class MyClass {  
    static sayHi() {  
        print("Hi");  
    }  
}  
  
main() {  
    MyClass.sayHi();  
}
```

# Linguagem de programação Dart

## Criando instância da Classe

- Para criar uma instância de classe, basta usar a palavra-chave **new** seguida do nome da classe.

- Sintaxe:

```
var object_name = new class_name([ arguments ]);
```

Ex.:

```
main() {  
    Hero h = new Hero ();  
    h.firstName = "Luke";  
    h.lastName = "Skywalker";  
    print(h.sayName());  
}
```

**Obs.:** No Dart, o new é opcional para instanciar classe.

Ex.: `var h = Hero();`



# Linguagem de programação Dart

## Construtores

- Função especial da classe que é executada para instanciar um objeto.
  - Ela informa como o objeto deve ser criado ou seja, informa como inicializar as variáveis da classe.

Ex.:

```
class Hero {  
  String? firstName;  
  String? lastName;  
  
  Hero(String fn, String ln) {  
    firstName = fn;  
    lastName = ln;  
  }  
  
  String sayName() {  
    return "$lastName, $firstName";  
  }  
}  
  
main() {  
  Hero h = new Hero("Luke", "Skywalker");  
  print(h.sayName());  
}
```

# Linguagem de programação Dart

## Construtores

- Dart fornece construtores nomeados para permitir que uma classe defina vários construtores.

- Sintaxe: `Class_name.constructor_name(param_list)`

Ex.:

```
void main() {  
    Car c1 = new Car.namedConst('E1001');  
    Car c2 = new Car();  
}  
class Car {  
    Car() {  
        print("Non-parameterized constructor invoked");  
    }  
    Car.namedConst(String engine) {  
        print("The engine is : ${engine}");  
    }  
}
```

# Linguagem de programação Dart

## Referência `this`

- A palavra-chave `this` referencia a instância atual da classe dentro da qual um bloco de código está sendo executado.

Ex.:

```
class Account {  
    int balance;  
  
    Account(int balance) {  
        this.balance = balance;  
    }  
}
```

# Linguagem de programação Dart

## Referência `this`

Ex.:

```
class Hero {
  String firstName;
  String lastName;

  Hero(this.firstName, this.lastName);

  String sayName() {
    return "$lastName, $firstName";
  }
}

main() {
  Hero h = new Hero("Luke", "Skywalker");
  print(h.sayName());
}
```

# Linguagem de programação Dart

## Subclasse (Herança)

- Uma classe pode herdar outra classe usando o comando **extends**;
  - Ela é chamada de subclasse.
- A palavra-chave **super** permite chamar métodos, acessar variáveis e o construtor na classe pai.
  - Sintaxe para o construtor da classe pai:

```
NomeDaSubclasse (par_1, par) : super (par_1)
```

- `par_1` representa os argumentos que serão passados para classe pai;
- `par` representa os argumentos que inicializaram a subclasse.

# Linguagem de programação Dart

## Subclasse (Herança)

Ex.:

```
class Hero {  
  String? firstName;  
  String? lastName;  
  
  Hero(this.firstName, this.lastName);  
  
  Hero.firstName(this.firstName);  
  
  String sayName() {  
    return "$lastName, $firstName";  
  }  
}
```

```
class UltimateHero extends Hero {  
  
  String? nickName;  
  
  UltimateHero(fn, ln, this.nickName) : super(fn, ln);  
  UltimateHero.build(fn) : super.firstName(fn);  
  
  String sayName() {  
    return "Nick name: $nickName | Jedi Master:  
$lastName, $firstName";  
  }  
}
```

# Linguagem de programação Dart

## Getters e Setters

- Getters e Setters, são chamados também de acessadores e mutadores;
- Eles permitem que o programa inicialize e recupere os valores dos campos de classe, respectivamente;
- Getters ou acessadores são definidos usando a palavra-chave `get`.
- Setters ou mutadores são definidos usando a palavra-chave `set`.
- Resumidamente, Getters e Setters são maneiras implícitas de implementar os métodos de recuperação e inicialização/atualização de valores.

# Linguagem de programação Dart

## Getters e Setters

- Sintaxe para definir um getter:

```
Return_type get identifier {  
}
```

- Sintaxe para definir um setter:

```
set identifier (parameter) {  
}
```



# Linguagem de programação Dart

## Getters e Setters

Ex.:

```
class Student {
  String name;
  int age;

  String get stud_name {
    return name;
  }

  void set stud_name(String name) {
    this.name = name;
  }

  void set stud_age(int age) {
    if(age<= 0) {
      print("Age should be greater
than 5");
    } else {
      this.age = age;
    }
  }

  int get stud_age {
    return age;
  }
}
```

```
void main() {
  Student s1 = new
  Student();
  s1.stud_name = 'MARK';
  s1.stud_age = 0;
  print(s1.stud_name);
  print(s1.stud_age);
}
```

# Linguagem de programação Dart

## Getters e Setters

Ex.:

```
class Hero {
  String? firstName;
  String? lastName;

  String get fullName => "$lastName,
$firstName";
  set fullName(n) => firstName = n;

  Hero(String fn, String ln) {
    firstName = fn;
    lastName = ln;
  }

  String sayName() {
    return "$lastName, $firstName";
  }
}
```

```
main() {
  Hero h = new Hero("Luke",
"Skywalker");

  print(h.sayName());
  print(h.fullName);

  h.fullName = "Anakin";

  print(h.fullName);
}
```

# Linguagem de programação Dart

## Interfaces

- Dart não possui uma sintaxe para declarar interfaces.
- As próprias declarações de classe são interfaces no Dart, ou seja, ele não faz distinção entre os conceitos de classe e interface.
- As classes devem usar a palavra-chave **implements** para poder usar uma interface (implementa-la).

- Sintaxe para implementação de uma interface:

```
class identifier implements interface_name{}
```

- Uma classe pode implementar várias interfaces.

- Sintaxe para implementação de várias interfaces:

```
class identifier implements interface_1, ..., interface_n {}
```

# Linguagem de programação Dart

## Interfaces – ex.:

```
class UltimateHero implements Hero {  
  @override  
  String? firstName;  
  @override  
  String? lastName;  
  
  UltimateHero(this.firstName, this.lastName);  
  
  @override  
  String get fullName {  
    return "$firstName this.lastName";  
  }  
  
  @override  
  set fullName(fn) {  
    firstName = fn;  
  }  
  
  String sayName() {  
    return "Jedi $lastName, $firstName";  
  }  
}
```

```
class Hero {  
  String? firstName;  
  String? lastName;  
  
  String get fullName => "$lastName, $firstName";  
  set fullName(n) => firstName = n;  
  
  Hero(String fn, String ln) {  
    firstName = fn;  
    lastName = ln;  
  }  
  
  String sayName() {  
    return "$lastName, $firstName";  
  }  
}  
  
main() {  
  Hero h = new UltimateHero ("Luke", "Skywalker");  
  
  print(h.sayName());  
  print(h.fullName);  
  
  h.fullName = "Anakin";  
  
  print(h.fullName);  
}
```

# Linguagem de programação Dart

## Interfaces – ex.:

```
void main() {  
    ConsolePrinter cp= new ConsolePrinter();  
    cp.print_data();  
}  
class Printer {  
    void print_data() {  
        print("_____Printing Data_____");  
    }  
}  
class ConsolePrinter implements Printer {  
    void print_data() {  
        print("_____Printing to Console_____");  
    }  
}
```

# Linguagem de programação Dart

Interfaces – ex.:

```
void main() {  
    Calculator c = new Calculator();  
    print("The gross total : ${c.ret_tot()}");  
    print("Discount :${c.ret_dis()}");  
}  
class Calculate_Total {  
    void ret_tot() {}  
}  
class Calculate_Discount {  
    void ret_dis() {}  
}  
class Calculator implements Calculate_Total,Calculate_Discount {  
    int ret_tot() {  
        return 1000;  
    }  
    int ret_dis() {  
        return 50;  
    }  
}
```

# Linguagem de programação Dart

## Classes Abstratas

- O modificador `abstract` são usado para definir classes abstratas que não podem ser instanciadas;
  - Sintaxe: `abstract class nameClass {}`
- Elas são usadas para definir interface disponibilizando métodos abstratos em seu escopo;
  - Métodos abstratos são usados para definir a assinatura de um método que será implementado por outra classe;
    - Sintaxe: `void identifier (par);`

# Linguagem de programação Dart

## Classes Abstratas

Ex.:

```
abstract class MyAbstractClass {  
  int x1, x2;  
  
  MyAbstractClass(this.x1, this.x2);  
  someMethod();  
}  
  
main() {  
  var obj = ImpMyAbstractClass(5, 6);  
  print(obj.someMethod());  
  print(obj.sum_);  
}
```

```
class ImpMyAbstractClass extends MyAbstractClass  
{  
  ImpMyAbstractClass(int a, int b) : super(a, b);  
  
  @override  
  String someMethod() {  
    return "This is example of abstract class!";  
  }  
  
  int get sum_ => x1 + x2;  
}
```



# Linguagem de programação Dart

## Exercícios

1. Crie um aplicativo para calcular a média aritmética das notas: 9,5; 7,0 e 4,0.
2. Crie um aplicativo que calcule a área de uma circunferência com raio = 3,5.
3. Crie um aplicativo para calcular o Índice de Massa Corpórea (IMC) de um indivíduo.  $IMC = \frac{Peso}{(Altura)^2}$
4. Imprima todos os números de 150 a 300.
5. Imprima a soma de 1 até 1000.
6. Imprima todos os múltiplos de 3, entre 1 e 100.
7. Imprima os fatoriais de 1 a 10.

# Linguagem de programação Dart

## Exercícios

9. Imprima os primeiros números da série de Fibonacci até passar de 100. A série de Fibonacci é a seguinte: 0, 1, 1, 2, 3, 5, 8, 13, 21, etc... Para calculá-la, o primeiro elemento vale 0, o segundo vale 1, daí por diante, o  $n$ -ésimo elemento vale o  $(n-1)$ -ésimo elemento somado ao  $(n-2)$ -ésimo elemento (ex:  $8 = 5 + 3$  ).
10. Escreva um programa que, dada uma variável  $x$  com algum valor inteiro, temos um novo  $x$  de acordo com a seguinte regra:
  - se  $x$  é par,  $x = x / 2$
  - se  $x$  é ímpar,  $x = 3 * x + 1$
  - imprime  $x$
  - O programa deve parar quando  $x$  tiver o valor final de 1. Por exemplo, para  $x = 13$ , a saída será:  $40 \rightarrow 20 \rightarrow 10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$

# Linguagem de programação Dart

## Exercícios

11. Imprima a seguinte tabela, usando fors encadeados:

```
1
2 4
3 6 9
4 8 12 16
n n*2 n*3 . . . . n*n
```

12. Crie uma classe “Forma” e será herdado pelas classe Quadrado, Trapézio e Circulo. A classe forma deve ser abstrata e possuir os seguintes métodos abstratos: calculo de área, e perímetro. Informações, métodos e dados complementares fica por cargo do desenvolvedor adicionar caso necessário. No entanto, as subclasses devem implementar os métodos abstratos.