SI700 – Programação para Dispositivos Móveis

Aula 1 – A Linguagem Dart

Prof. Ulisses Martins Dias

2023

Faculdade de Tecnologia – Unicamp

Roteiro

Introdução

Introdução

- A linguagem Dart é orientada a objetos.
- O estilo é baseado em C.
- A linguagem é tipada, havendo opção de tipagem estática e dinâmica.
- Os comentários seguem o padrão do Java (single e multiline). Uma sintaxe de comentários para documentação também existe.

```
// Isto é um comentário

/*
   Um comentário de
   múltiplas linhas
*/

/// Comentário de documentação

/**
  Comentário de documentação
  com múltiplas linhas
*/
```

- A linguagem descreve tanto o comportamento quanto a interface gráfica dos aplicativos.
- Uma função main é o ponto de partida do código.

```
void main() {
  // Escreva algum código
}
```

Tipos Básicos

Tipos Básicos

- Em Dart, declarar o tipo é uma opção.

```dart
String name = "Ulisses";
int idade = 25;
double altura = 1.74;
bool casado = true;
```

- Outra opção é declarar var.

```
var name = "Ulisses";
var idade = 25;
var altura = 1.74;
var casado = true;
```

- Para obter o tipo da variável em tempo de execução, use o comando runtimeType.
- Para saber se uma variável é de um dado tipo, use o comando is.

```
var altura = 1.74;
print(altura is int); // false

print(altura.runtimeType);// double
```

- Em Dart, você poderá mudar o tipo de uma variável declarada como dinâmica (dynamic).

```
dynamic altura = 1.74;
print(altura.runtimeType);// double

altura = "gigante";
print(altura.runtimeType);// String

altura = true;
print(altura.runtimeType);// bool
```

- A palavra reservada const declara constantes conhecidas em tempo de compilação.
- A palavra reservada final declara constantes cujo valor só pode ser associado uma vez (em tempo de compilação ou execução).
- Em ambos, a instanciação do valor é obrigatória.

```
// Um valor deve ser atribuído
// em tempo de compilação
const altura = 1.74;

// O valor da constante abaixo
// pode ser atribuído em tempo
// de execução.
final agora = DateTime.now();
```

- As conversões para string usam toString, as conversões para tipos numéricos usam parse.

```
int idade = 25;
double altura = 1.74;
String si = idade.toString();
String sa = altura.toString();

int id1 = int.parse(si);
double al1 = double.parse(sa);
print("Idade: $id1, Altura: $al1");
// Idade = 25, Altura = 1.74
```

Tipos Estruturados

Tipos Estruturados

- Os principais tipos estruturados são sequências (List), dicionários (Map) e conjuntos (Set). Para cada tipo, existe uma gama de funções.

- **Sequências**

```
/* List: sequência de valores indexáveis
 pela posição. Podemos mudar valores
 existentes e acrescentar novos. */
var seq = ["a", "e", "i", 1, 2];
String k = seq[2]; // k recebe "i"
print(seq.runtimeType);//JSArray<Object>

seq.add(3);
print(seq); //[a, e, i, 1, 2, 3]
print(seq.indexOf("e")); // 1
```

```
// Podemos iterar com o método forEach
 seq.forEach(print);
/*
a
e
i
1
2
3
*/
```

```
// Map: Pares "chave : valor"
var dic = {
    "key"   : "value",
    1       : "one",
    3.14    : "pi",
    "flag"  : true
  };
print(dic);
/* {key: value, 1: one,
   3.14: pi, flag: true} */
```

Tipos Estruturados - Dicionários

```
var x = dic["key"]; // Acessos
print(dic.runtimeType);
//JsLinkedHashMap<Object,Object>

// Acrescentando novos elementos
dic[2]      = "dois";
dic["dois"] = 2;
print(dic);
/*
{key: value, 1: one, 3.14: pi,
flag: true, 2: dois, dois: 2}
*/
```

```
// Podemos iterar com a função forEach
dic.forEach( (key, val) {
    print("C: $key, V: $val");
});
/*
C: key, V: value
C: 1, V: one
C: 3.14, V: pi
C: flag, V: true
C: 2, V: dois
C: dois, V: 2
*/
```

Tipos Estruturados - Dicionários

```dart
var docentes = Map<String, int>();
docentes["Ulisses"] = 5;
docentes["Meira"] = 3;
docentes["Marco"] = 1;
docentes["Gisele"] = 4;

print(docentes);
/*
{Ulisses: 5, Meira: 3,
Marco: 1, Gisele: 4} */

docentes.remove("Marco");
print(docentes);
// {Ulisses: 5, Meira: 3, Gisele: 4}
```

```
/*
  Podemos também obter as chaves e
  os valores separadamente.
*/

print(docentes.keys);
//(Ulisses, Meira, Gisele)

print(docentes.values);  //(5, 3, 4)
```

Tipos Estruturados - Dicionários

```
Map discentes = { };
print(discentes.isEmpty); // true

discentes["Bernini"] = 2;
discentes["Gislaine"] = 3;

discentes.forEach((k, v) {
    print( k +" discente #" +
      v.toString());
});
/*
Bernini discente #2
Gislaine discente #3
*/
```

```
/*
 Set: itens não ordenados dentro do
conjunto e não há elemento repetido
*/

Set docentes = Set();
docentes.addAll([ "Ulisses",
    "Meira", "Leon", "Ulisses"]);
docentes.add("Ana Estela");
docentes.remove("Meira");

print(docentes);
// {Ulisses, Leon, Ana Estela}
```

```
print(docentes.contains("Ulisses"));
// true

print(docentes.containsAll(
    [ "Meira",
      "Ana Estela" ]
)); // false

/* A linha a seguir gera ERRO */
print(docentes[0]);
// Não é possível indexar Set
```

Tipos Anuláveis

Tipos Anuláveis

- Os tipos que vimos até agora eram não anuláveis. Isso quer dizer que não podemos atribuir o valor nulo a eles e que não podemos usar sem a primeira atribuição.

```
/* A value of type 'Null' can't be assigned to a variable
   of type 'int'. */
int value = null;

/* The non-nullable local variable 'value' must be assigned
   before it can be used. */
int value;
print(value);
```

Tipos Anuláveis

- Uma variável não inicializada é automaticamente definida como nula, o que seria um erro. A fim de compilar com sucesso, inicialize a variável logo que for declarada:

```
// Caso 1.
int value = 0;
print("$value");

// Caso 2.
int value;
value = 0;
print("$value");
```

Tipos Anuláveis

- Não faça verificações de valores nulos em variáveis de tipos não anuláveis porque é inútil.

```
String name = "Ulisses";
void main() {
  if (name != null) { // Inútil
     print(name)
  }
}
```

Tipos Anuláveis

- É também possível declarar tipos anuláveis que não necessitam ser inicializados antes de serem usados e, portanto, é permitido serem nulos.
- Se você anexar um ponto de interrogação no final do tipo, você recebe um tipo anulável.

```
int? value;
print("$value"); // Linha válida que imprime null
```

Tipos Anuláveis

- Os tipos anuláveis que suportam o operador de índice [] precisam ser chamados com a sintaxe ?[]. O valor nulo é retornado se a variável também for nula.

```
List<String>? names1 = ["Ulisses", "Marco", "Ana Estela"];
String? first1 = names1?[0];
print(first1); // Ulisses

List<String>? names2;
String? first2 = names2?[0];
print(first2); // null
```

Tipos Anuláveis

- Observe que o operador de . não está disponível em tipos anuláveis, ao invés disso, você deverá usar o operador ?. para fazer acesso aos membros.

```
double? pi;
final round = pi?.round(); // Ok
print(round);

/*
A linha a seguir geraria um erro.

final round = pi.round();
*/
```

Tipos Anuláveis

- Quanto temos certeza de que um variável de tipo anulável não é nula, podemos usar o operador ! para convertê-la em uma versão não anulável.

```
// O código a seguir é possível.
int? nullable = 0;
int notNullable = nullable!;

// O código a seguir gera um erro.
int? nullable;
int notNullable = nullable!;
```

Tipos Anuláveis

- Quando queremos converter uma variável de tipo anulável em um subtipo não anulável, devemos usar o operador as. O operador ficará mais claro quando entendermos de orientação a objetos em Dart.
- Note no exemplo a seguir que não poderíamos ter usado o operador !, dada a necessidade de conversão de tipo.

```dart
num? value = 5;
int otherValue = value as int;
print(otherValue); // 5
```

Operadores

Operadores Aritméticos

```dart
double a = 23.0;
double b = 7.0;

a  +  b; // Adição :         30.0
a  -  b; // Subtração:       16.0
a  *  b; // Multiplicação:   161.0
a  /  b; // Divisão: 3.2857142857
a  ~/ b; // Divisão Inteira: 3
a  %  b; // Resto da Divisão: 2
```

Operadores Aritméticos

```
/*
 Abaixo, o primeiro valor impresso é 23,  mas o valor
 é incrementado logo em seguida
*/
print(a++); // 23
print(a);   // 24

/*
 Abaixo, o valor impresso é 25,  pois o incremento
 ocorreu antes de retornar o valor da expressão
*/
print(++a); // 25
print(a);   // 25
```

Operadores de Atribuição

```
/*
  O incremento pode ocorrer simultaneamente com
  a atribuição.
*/

double a = 23.0;

a += 1; // 24
a -= 1; // 23
a *= 2; // 46
a /= 2; // 23
```

Operadores de Comparação

```
double a = 23.0;
double b = 7.0;

a  <  b; // Menor que:       false
a <=  b; // Menor ou igual: false
a ==  b; // igual:          false
a  >  b; // Maior que       true
a >=  b; // Maior ou igual  true
a !=  b; // Diferente       true
```

Operadores Lógicos

```
bool a = true;
bool b = false;

// Operadores Lógicos
a   &&   b;  // false
a   ||   b;  // true
!   a;       // false
```

Comandos Condicionais

- O comando if só aceita resultados booleanos. A ideia comum em outras linguagens de que existe um "contexto" booleano não é válida. Ou seja, o inteiro 1 não será considerado verdadeiro e nem o inteiro 0 será considerado falso.

```
var professor = "Ulisses";
if (professor=="Ulisses" || professor=="Meira") {
  print ("Professor FT/Unicamp");
} else if (professor=="Zanoni") {
  print ("Professor IC/Unicamp");
} else {
  print("Não sei quem é");
}
```

- O comando switch pode lidar com tipos não booleanos, desde que os objetos comparados sejam do mesmo tipo (subclasses não são permitidas) e as classes não sobrescrevam o operador ==.
- É comum o comando switch ser usado com enumerate. Neste caso, um erro será gerado se faltar cláusula para algum dos elementos no enumerate.

Comandos Condicionais

```
enum Disciplinas {SI700, SI202, SI101, SI100}
var disciplina = Disciplinas.SI700;
switch(disciplina) {
  case Disciplinas.SI700 :
     print("Ambos os semestres");
     break;
  case Disciplinas.SI202 :
     print("Segundo semestre");
     break;
  case Disciplinas.SI101 :
  case Disciplinas.SI100 :
     print("Primeiro semestre");
     break;
} // Ambos os semestres
```

Comandos Condicionais

- Existem operadores ? e ?? para gerar comandos condicionais com apenas uma linha de código.

```
// Condição ternária:
bool a = true;
int b = 1;
int c = 2;
var d;

// Se a for verdadeiro, então retorna b, caso contrário c.
print(a? b : c); // 1

/* Se d for não nulo retorna d, caso contrário b */
print(d ?? b); // 1
```

- O operador ?? pode ser usado para gerar valores que podem ser atribuídos diretamente a variáveis não anuláveis a partir de variáveis de tipos anuláveis.

```
int? nullable = 10;
int nonNullable = nullable ?? 0;
print(nonNullable); // 10
```

Laços de Repetição

Comandos Condicionais

- Os laços de repetição são muito semelhantes ao que encontramos na linguagem Java. Vamos exemplificar os principais usos dos laços for, while e do ... while.
- **Comando while**

```
int count  = 0;
while (count < 4) {
  print("Count = $count");
  count = count + 1;
}
/*
Count = 0
Count = 1
Count = 2
Count = 3
*/
```

- **Comando do ... while**

```
int count  = 0;
do {
  print("Count = $count");
  count = count + 1;
  if (count == 2) {
    break;
  }
} while (count < 5);
/*
Count = 0
Count = 1
*/
```

- **Comando for**

```
var soma = 0;
for (var i =1; i <= 10; i++){
  soma += i;
}
print(soma); // 55

// Iteração sobre iterators
var numeros = [1, 2, 3, 4, 5];
for (var  num in numeros){
  soma += num;
}
print(soma); // 70
```

Funções

Funções - Forma Básica

- Possuem nome, corpo onde se coloca códico, lista de parâmentros de entrada e um tipo de retorno.

```
// Esta função retorn null
void hello_world() {
  print("Hello World!");
}
// Esta função recebe um parâmetro
void hello_user(user) {
  print("Hello $user");
}
void main() {
  hello_world(); // Hello World!
  hello_user("Ulisses");//Hello Ulisses
}
```

- A declaração de tipo de retorno é opcional. Se o programador não declarar uma cláusula return, então a função retorna null.

```
findArea(int altura, int largura){
  return altura * largura;
}

main() {
  int x = findArea(2,4);
  print(x); // 8
}
```

Funções - Usando como Expressões

- Sintaxe mais simples com o operador => para apenas um comando.

```
int findArea(int altura, int largura) =>
  altura * largura;

main() {
  int x = findArea(2,4);
  print(x); // 8
}
```

Funções - Usando como Variáveis

- Funções podem ser passadas como parâmetros ou atribuídas a variáveis.

```
// Função que soma
int soma(a, b) {
  return a+b;
}
// Função  depende de outra passada como parâmetro.
doSomething(param_a, param_b, funcao){
  return funcao(param_a, param_b);
}

void main() {
  var x = doSomething(2, 5, soma);
  print(x);
}
```

Funções - Passagem de Parâmetros

Parâmetros Opcionais Posicionais

- Parâmetros podem ser declarados opcionais com colchetes. A ordem dos parâmetros define como os valores passados pelo usuário serão atribuidos.
- Se o usuário decidir não passar valor naquela posição, então o valor null será usado por padrão.

Parâmetros Opcionais Posicionais

```
//  Argumentos posicionais opcionais: colocar após os obrigatórios
void hello_familia(user, [esposa]) {
  print("Hello $user e $esposa");
}
// Função Principal
void main() {
  // Argumentos opcionais posicionais
  hello_familia("Ulisses");
  hello_familia("Ulisses", "Danielle");
}
/*
Hello Ulisses e null
Hello Ulisses e Danielle
*/
```

Parâmetros Opcionais Nomeados

- Uma chave ao redor de um parâmetro também indica que são opcionais. Entretanto, neste caso, o programador deverá fornecer o nome do parâmetro ao qual gostaria de fornecer um valor.
- Note especialmente que a ordem em que os parâmetros são declarados na assinatura da função não mais importa.

Parâmetros Opcionais Nomeados

```
void hello_amigos(String user, {String esposa, String amiga}){
   print("Hello $user, $esposa e $amiga");
}
void main() {
  // Argumentos opcionais nomeados
  hello_amigos("Ulisses");
  hello_amigos("Ulisses",esposa:"Dani");
  hello_amigos("Ulisses", esposa:"Dani", amiga:"Pri");
  hello_amigos("Ulisses", amiga:"Pri", esposa:"Dani");
}
/* Hello Ulisses, null e null
   Hello Ulisses, Dani e null
   Hello Ulisses, Dani e Pri
   Hello Ulisses, Dani e Pri   */
```

Parâmetros com Valores Default

- Nos casos dos parâmetros opcionais vistos acima, um valor null é atribuído quando nada é fornecido na invocação. Entretanto, a própria função pode ter um valor default para esses casos.
- O valor default é informado no momento da declaração do parâmetro opcional dentro das chaves.

Parâmetros com Valores Default

```
void hello_all(String u,{String esposa="Dani",  String amiga="Pri",
      String cachorro = "Snoop", String gato = "Nini" } ) {
  print("Hello $u, $esposa, $amiga, $cachorro, $gato");
}
void main() {
  hello_all("Ulisses");
  hello_all("FT", amiga:"Mari");
  hello_all("FT", cachorro:"Boró");
  hello_all("FT", gato:"Mingau");
}
/*  Hello Ulisses, Dani, Pri, Snoop, Nini
    Hello FT, Dani, Mari, Snoop, Nini
    Hello FT, Dani, Pri, Boró, Nini
    Hello FT, Dani, Pri, Snoop, Mingau       */
```