DART

Daniel Duque
Gustavo Alochio
Lorenzo Moulin
Usiel Lopes

INTRODUÇÃO

HISTÓRIA E CRIAÇÃO

- Desenvolvida pelo Google. Iniciada por Lars Bak, Google Chrome (V8) e por Kasper Lund
- Concebida na conferência GOTO na Dinamarca em outubro de 2011
- Criada com a proposta de deixar o JavaScript e Java

Google

MERCADO E O HYPE

- Linguagem voltada para Web
- Flutter Framework

Dart > Flutter > Apps

- Desenvolvimento Híbrido
  - Android
  - iOS
  - Web

- Fuchsia SO

A LINGUAGEM

- Open-source, Licença: BSD.

- Influenciada por CoffeScript, Go, Java, JavaScript e C++.

- Programas nesta linguagem podem tanto serem executados em uma máquina virtual (Dart VM) quanto compilados para JavaScript.

A LINGUAGEM

- O SDK também inclui o utilitário -dart2js que é um tradutor que gera o código JavaScript equivalente ao do Dart Script.
- Multiparadigma.
- Última versão: 2.6.0 (27 de junho de 2019)

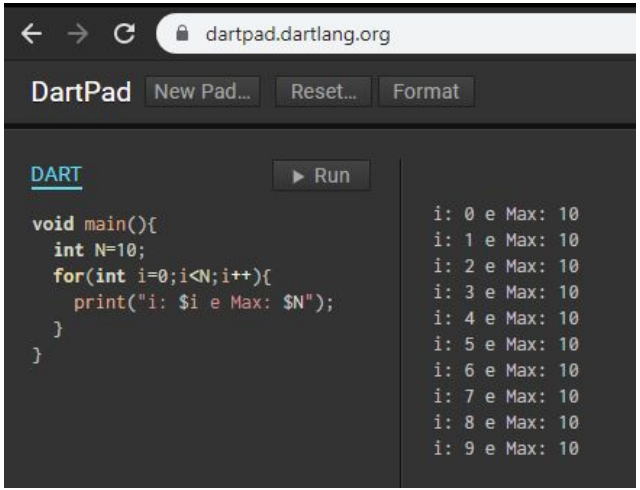
A LINGUAGEM

- Tudo o que se pode colocar em uma variável é um objeto e todos os objetos são herdados da classe Object.

- Suporta uma variedade de auxiliadores: interfaces, classes, coleções, funções de nível superior, funções aninhadas, tipos genéricos e tipagem opcional.

Suporte para desenvolvimento

- DartPad - Online
  - https://dartpad.dartlang.org/
  - Para aprendizado
- Eclipse
- IntelliJ
- WebStorm
- Visual Studio Code



Instalação e Como usar ?

- Seguir documentação para instalação
- Extensão do arquivo : .dart
- Execução do programa:
  - Checked Mode: “-c” ou “--checked”. Ex: ***dart -c Test.dart***
    - Adiciona warning, erros para auxiliar o desenvolvimento e o debugador.
    - Reforça várias verificações como a verificação de tipo, entre outras.
  - Production Mode: Default. Ex: ***dart Test.dart***
    - Bom para garantir desempenho.

Comentários

- `// Comentário na mesma linha.`
- ```
  /*
  Comentário
  em
  várias
  linhas.
  */
  ```
  ```

Espaços em branco, quebra de linha, ponto e vírgula

- Dart ignora espaços, tabs, linhas novas que aparecem no programa.
- Pode usar a vontade, de preferência de uma maneira que o código fique bem legível.
- Ponto e vírgula (;) é usado para a terminação de *statements*.

Exemplo de código

```
// Definição da função
imprimirInteiro(int umNumero) {
  print('O numero é $umNumero.'); // imprime no console
}

//Aqui é onde começa executando
main() {
  var numero = 42; // Declara e inicializa a variável
  imprimirInteiro(numero); // Chama a função
}
```

- **Saída**

```
O numero é 42.
```

AMARRAÇÕES

AMARRAÇÕES - PALAVRAS RESERVADAS

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

Identificadores

São os nomes dados aos elementos no programa, como: variáveis, funções, etc..

- Não pode começar com um dígito, mas pode conter dígito.
- Não podem 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.

Identificadores - Continuação

- Contrariando Java, Dart não utiliza palavras-chave public, protected e private.
- Em Dart se um identificador começa com um sublinhado (_), é privado para sua biblioteca.

```
library other;

class A{

    int _private;

}
```

Tempos de amarração

- Identificador “var” faz inferência de tipos em tempo de compilação.

  ```
  var i = “test”;
  ```

Ambiente e escopo de amarração

- Escopo aninhado e estático

Ambiente e escopo de amarração - exemplo

```
main() {
  int x = 0;
  print("$x");
  if(x==0){
    int x = 1;
    print("$x");
    x++;
    print("$x");
  }
  print("$x");
}
```

```
0
1
2
0
```

Definição e Declaração

Tipos de Definição:

var:

```
var i = 0;
```

const ou final :

```
const fixo = 10;
final String nome = “Vitor”;
```

Tipo :

```
int i = 0;
```

VALORES E TIPOS DE DADOS

Tipagem

Dart é fortemente tipado, porém anotações de tipo são opcionais, pois o Dart pode inferir tipos. Se não quiser explicitar um tipo, use o identificador var para inferir o tipo, ou se quiser modificar o tipo do conteúdo da variável, utilize o tipo dynamic.

Ex:

```
bool convertToBool(dynamic arg) {
  if (arg is bool) return arg;
  if (arg is String) return arg == 'true';
  throw ArgumentError('Cannot convert $arg to a bool.');
}
```

Tipos incorporados para linguagem

- Não existem tipos primitivos
- numbers
- strings
- booleans
- lists (também conhecido como array)
- sets
- maps
- runes (para expressar caracteres Unicode na string)
- symbols

NUM CLASS

A classe num pode ser uma das classes int ou double.

Se qualquer outra classe tentar herdar ou implementar esta classe, é gerado um erro em tempo de compilação.

Numbers

Integer

Valores Integer possuem 64 bits usando a notação de complemento de dois.

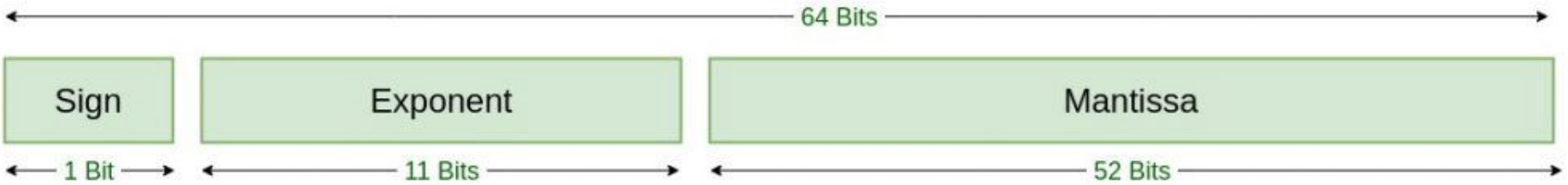
Dart VM: -2^63 até 2^63-1

Dart que é compilado para JavaScript usa os números de JavaScript: -2^53 até 2^53-1

Numbers - Continuação

Double

É número de ponto flutuante de precisão dupla (64 bits) como especificado pela IEEE 754



Double Precision
IEEE 754 Floating-Point Standard

Numbers - exemplo

Integer - keyword: int

```dart
int oito = 8;
print(oito); //8
```

Double - keyword: double

```dart
double novePontoDois = 9.2;
print(novePontoDois);
```

String

String em Dart é uma sequência de unidades do código UTF-16.

Pode usar tanto aspas simples (‘’) quanto aspas duplas (“”) para criar strings.

String - Exemplo

```dart
String FRASE = "  ESTOU NO SEMINÁRIO DE DART";
print(FRASE);

String frase = FRASE.toLowerCase();

print(frase);
print(frase.trim());
```

```
  ESTOU NO SEMINÁRIO DE DART
  estou no seminário de dart
estou no seminário de dart
```

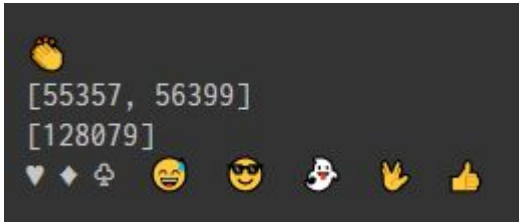
Runes

O unicode define um sistema único de valores para cada letra, dígito e símbolo.

Por isso, para expressar 32-bit unicode valores com uma string, é necessário uma sintaxe especial.

Runes - Exemplo

```
var clapping = '\u{1f44f}';
print(clapping);
print(clapping.codeUnits);
print(clapping.runes.toList());
```




```
Runes input = new Runes(
  '\u2665 \u2666 \u2667  \u{1f605}  \u{1f60e}  \u{1f47b}  \u{1f596}  \u{1f44d}');
print(new String.fromCharCodes(input));
```

Booleans

Tipo boolean só pode ser true ou false

keyword = bool

```
bool toggle = false;
print(toggle);
if(toggle)
  toggle = false;
else{
  toggle = true;
}
print(toggle);
```

```
false
true
```

Enumerado

```dart
enum Boleano {
  verdadeiro,
  falso,
}

void main() {
   print(Boleano.values);
   // [Boleano.verdadeiro, Boleano.falso]
   Boleano.values.forEach((v) => print('value: $v, index: ${v.index}'));
   // value: Boleano.verdadeiro, index: 0
   // value: Boleano.falso, index: 1
   print('Testando: ${Boleano.verdadeiro}, ${Boleano.verdadeiro.index}');
   // Testando: Boleano.verdadeiro, 0
   print('Testando index: ${Boleano.values[1]}');
   // Testando index: Boleano.falso
}
```

Lists

Tipo lista é usado para representar coleção de objetos.

A lista é um grupo indexado de objetos. (índices 0...n-1)

Lista em Dart é sinônimo do conceito de array em outras linguagens de programação.

Lists - exemplo

```
List<int> fixedLengthList = new List(5);

fixedLengthList.length = 0;  // Error

fixedLengthList.add(499);    // Error

fixedLengthList[0] = 87;

List<int> growableList = [1, 2];

growableList.length = 0;

growableList.add(499);

growableList[0] = 87;
```

Sets

É uma coleção não ordenada, diferente de uma lista.

O tipo Set sempre esteve no Dart, mas a partir da versão 2.2 foi introduzido o Set literal que é similar ao Map.

Sets - Exemplo

```dart
var halogens = {'fluorine', 'chlorine', 'bromine', 'iodine', 'astatine'};
```

Dart infere que halogens tem o tipo Set<String>. Se tentar adicionar um tipo diferente de String, é gerado um erro.

```dart
var names = <String>{};
Set<String> names = {}; // Isso funciona também.
var names = {}; // Cria um mapa.. não um Set.
```

Bem fácil de confundir com o mapa na hora de iniciar a variável.

SplayTreeSet

A mesma ideia do Set, só que a estrutura mantém uma ordenação dos elementos. Operações em tempo logarítmico amortizado.

```
SplayTreeSet<int> s = SplayTreeSet();

print(s.contains(2)); //false

s.add(3); //O(logn)

s.add(2); //O(logn)

s.add(1); //O(logn)

s.forEach((e) => print(e)); // 1 2 3
```

Maps

O tipo mapa representa um conjunto de valores e suas respectivas chaves.

Tem métodos bem definidos para acessar suas chaves e respectivos valores.

Parecido com Dictionary de Python.

Maps - Exemplo

```dart
Map<String,dynamic> infosPessoa = Map<String,dynamic>();
infosPessoa = {
  "Nome": "Ronaldo",
  "Idade": 50,
  "Endereço": "Rua..",
  "Filhos": ["Filho1","Filho2","Filho3"]
};
print(infosPessoa);
print(infosPessoa.keys);
print(infosPessoa.values);
```

```
{Nome: Ronaldo, Idade: 50, Endereço: Rua.., Filhos: [Filho1, Filho2, Filho3]}
(Nome, Idade, Endereço, Filhos)
(Ronaldo, 50, Rua.., [Filho1, Filho2, Filho3])
```

SplayTreeMap

A mesma ideia do Map, só que a estrutura mantém uma ordenação dos elementos. Operações em tempo logarítmico amortizado.

```
SplayTreeMap<String,dynamic> infosPessoa = SplayTreeMap<String,dynamic>();

infosPessoa["Nome"] = "Ronaldo";

infosPessoa["Endereço"] = "Rua...";

infosPessoa["Filhos"] = ["Filho1","Filho2","Filho3"];

print(infosPessoa); //{Endereço: Rua..., Filhos: [Filho1, Filho2, Filho3], Nome: Ronaldo}

print(infosPessoa.keys); //(Endereço, Filhos, Nome)

print(infosPessoa.values); //(Rua..., [Filho1, Filho2, Filho3], Ronaldo)
```

Dynamic

Pode mudar o tipo conforme a variável é utilizada.

```
dynamic v;
v = true;
print(v);
if(v)
  v = 3;
print(v);
if(v == 3)
  v = "tres";
print(v);
v = [1,"dois",true];
print(v);
```

```
true
3
tres
[1, dois, true]
```

Var

Infere o tipo pela inicialização da variável.

Depois da inferência do tipo, a variável só é daquele tipo.

```
var nome = 'nome';
if(nome == 'nome')
  print("sou uma string");

var num = 1;
if(num == 1)
  print("${num*3}");
```

```
sou uma string
3
```

Erros de Tipo

Dependendo da versão do Dart, podem ocorrer erros com determinados tipos.

Erros de tipos comuns em Dart?

Erros de Tipo - Versão

Antes do Dart 2.1

```dart
double num = 1; // Erro em usar integer em um contexto de double.
                // Cuidado com o escopo de num.
```

Depois do Dart 2.1

```dart
double num = 1; // Equivalente a double num = 1.0.
```

Erros de Tipo - Dynamic vs Var

```dart
var souString = 'sou uma string por que var infere isso';
print (souString ); // sou uma string por que var infere isso
souString = 3; //
```

error	A value of type 'int' can't be assigned to a variable of type 'String'.

```dart
dynamic dinamico = 'sou dinamico';
print (dinamico); // sou dinamico
dinamico = 3;
print(dinamico); // 3
```

```
sou dinamico
3
```

VARIÁVEIS E CONSTANTES

Variáveis

Regra dos identificadores mostrado anteriormente.

Todas variáveis são iniciadas com null, porque Dart considera todos valores como objetos.

```
var souNull;
print(souNull); //null
```

Final e Const

Final e Const são usados para declarar constantes. Dart não deixa modificar o valor!

Pode usar em conjunto com o tipo da variável ou no lugar do “var”.

Const - Constante em tempo de compilação. Variáveis declaradas usando “const” são implicitamente “final”.

Final - Constante em tempo de execução, por exemplo.. algum request do server que você não quer que seja modificado.

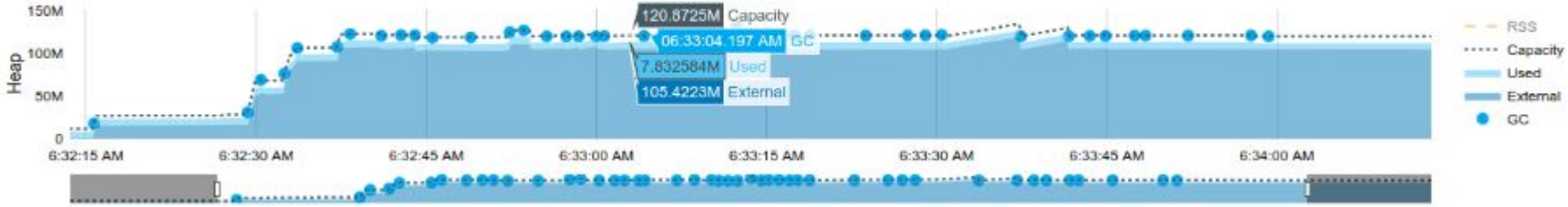
Final e Const - Exemplo

```dart
int a = 3;
int b = 4;

final int fSoma = a + b; // 7
const int cSoma = a + b; // Error const variables must be initialized with a constant value
const int cSoma = 7; // 7

fSoma = 42; // 'fSoma', a final variable, can only be set once.
```

Pilha - Heap

Objetos em Dart criados usando os construtores de suas classes ficam na pilha(heap).

Dart DevTools Memory - é possível ver partes isoladas da memória em determinados momentos para saber se há vazamento de memória ou para saber o ritmo de alocação de memória.



Gc - quando o garbage collector passou.
Used - objetos na heap

I/O e Serialização

Serialização padrão do Dart não existe.

Existem bibliotecas feitas para serialização feitas pela comunidade e podem ser usadas no Flutter.

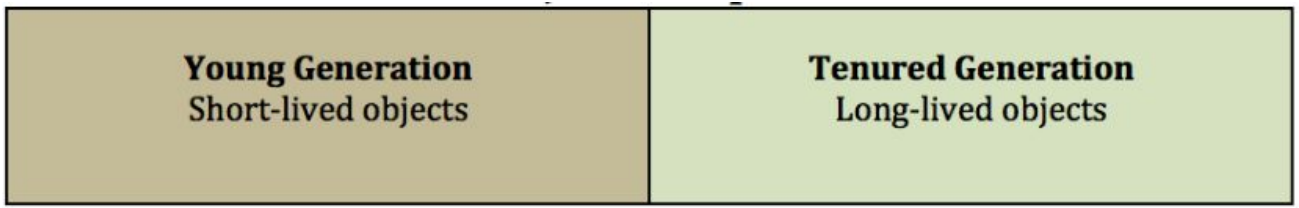
Para I/O é usado a biblioteca - `import 'dart:io';`

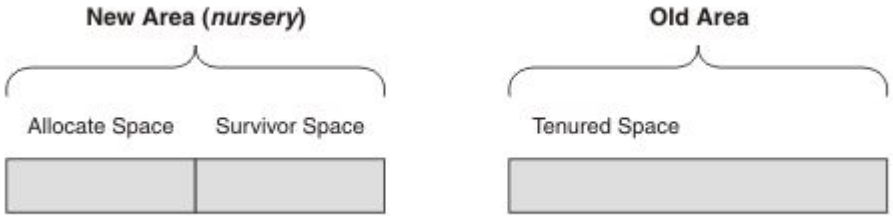
Essa biblioteca trabalha com arquivos, diretórios, processos, HTTP servers, clientes e outras coisas.

I/O - Exemplo

```dart
Future main() async {
  var config = File('config.txt');
  var conteudo;

  // Coloca o arquivo inteiro em uma string
  conteudo = await config.readAsString();
  print('The file is ${conteudo.length} characters long.');

  // Lê linha a linha
  conteudo = await config.readAsLines();
  print('The file is ${conteudo.length} lines long.');
}
```

Coletor de Lixo - Geracional





Coletor de Lixo - The young space scavenger

"O jovem limpador de espaços"

É feita para limpar objetos efêmeros (que tem vida útil curta).

Quando objetos atingem certa vida útil, são promovidos para um novo espaço de memória.

Coletor de Lixo - Parallel mark sweep collectors

“Coletores de varredura de marca paralela”

Também conhecido como marcar-varrer. Essa técnica tem duas fases.

1. Os objetos são percorridos pela primeira vez e os que ainda estão sendo usados são marcados.
2. A memória inteira é varrida e qualquer objeto que não estiver marcado é reciclado. Depois disso, todas as marcações são limpas e o ciclo recomeça.

EXPRESSÕES E COMANDOS

Operadores Aritméticos

Operador	Significado	Exemplo
+	Soma	3 + 2 = 5
-	Subtração	3 - 2 = 5
*	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	i = 3; i++; // 4
--	decrementar em 1	i = 3; i--; // 2

Operadores de Igualdade e Relacional

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

Operadores de Atribuição

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

Operadores Lógicos e Verificação de tipo

Operador	Significado	Exemplo
&&	AND - Retorna true se as duas expressões forem true	true && true = true
||	OR - Retorna true se pelo menos uma expressão for true	true || false = true
!	NOT - nega o resultado da expressão	!false = true

is	True se o objeto tem o tipo especificado	int a; a is int // true
is!	False se o objeto tem o tipo especificado	int a; a is! int // false

Operadores Bit a Bit

Operador	Significado
a & b	AND bit a bit
a | b	OR bit a bit
a ^ b	XOR bit a bit
~ a	NOT bit a bit
a << b	Deslocamento para esquerda
a >> b	Deslocamento para esquerda

Decisão - if, else if, else

```
if (x > 10) {
    print(true);
} else if (x < 10) {
    print(false);
} else {
    print(10);
}
```

Loop - FOR, FOR-IN

FOR

```
for (var i = 0; i < 3; i++) {
  print(i); 0 1 2
}
```

FOR-IN

```
var lista = [0,1,2];
for (var x in lista) {
  print(x); // 0 1 2
}
```

Loop - while, do-while

while

```
int i = 10;
while(i>0){
    print(i);
    i--;
}
```

do-while

```
int i = 10;
do{
    print(i);
    i--;
}while(i>0);
```

Break

Break - Usado para acabar o loop

```
int i = 0;
while(true){
  i++;
  print(i); // 1 2
  if(i==2)
    break;
}
```

Continue

Continue - Usado para avançar para o próximo loop.

```
for(int i=0; i < 10;i++){
   if(i % 2 == 0)
     continue;
   print(i); // 1 3 5 7 9
 }
```

Switch e Case

Cada “case” é obrigatório terminar com um dos seguintes statements: break, continue, rethrow, return, throw.

```
var comando = 'NADA';
switch (comando) {
case 'FECHADO':
  print("Fechado");
  break;
case 'ABERTO':
  print("Aberto");
  break;
default:
  print("Executa se não entrar em nenhum case"); // Vai executar esse
}
```

Assert

Usado durante o desenvolvimento, ele interrompe a execução normal se a condição do Assert for falso.

Quando o Dart é executado no production mode (default), Asserts são ignorados.

Quando Assert é avaliado?

Flutter - quando está no debug mode.
Dart - ao executar a linha de comando com a flag “--enable-asserts”.

Assert - Exemplo

```dart
var text; // null
var number = 9;
String urlString = "https://www.google.com.br/";

// Certifica-se que o valor não é nulo
assert(text != null, "Seu texto é nulo"); // Assertion error - Seu texto é nulo

// Certifica-se que o valor é menos do que 100
assert(number < 100, "Você colocou um numero maior do que 100"); // Continua o código

// Certifica-se que a string começa com https
assert(urlString.startsWith('https'), "Não começa com https"); // Continua o código
```

Expressões Condicionais

condition ? expr1 : expr2

Se ***condition*** é ***true*** avalia e retorna ***expr1***, caso contrário avalia e retorna ***expr2***.

```
var i = 2;
i == 1 ? print("é 1") : print("não é 1");
```

vs

```
if (i == 1) {
  print("é 1");
} else{
  print("não é 1");
}
```

Expressões Condicionais - Continuação

`expr1 ?? expr2`

Se `expr1` é `null` retorna `expr2`, caso contrário avalia e retorna `expr1`.

```
var n1;
print(n1 ?? "souNulo"); // souNulo
```

vs

```
if (n1 == null) {
  print("souNulo"); // souNulo
} else {
  print(n1);
}
```

MODULARIZAÇÃO

Funções

```dart
int soma1(int a,int b){
  return a + b;
}

int soma2(int a,int b) => a + b; // Lambda

void main(){
  int i = 1;
  int j = 2;
  print(soma1(i,j)); // 3
  print(soma2(i,j)); // 3
}
```

Função Anônima

```
void main(){
  var list = ['apples', 'bananas', 'oranges'];

  list.forEach((item) {
    print('${list.indexOf(item)}: $item');
  });
}
```

```
0: apples
1: bananas
2: oranges
```

Parâmetros - Opcional e Default

```dart
void printar1({int a, int b, int c = 3}){ // Opcional nomeado
  print("a = $a, b = $b, c = $c");
}
void printar2([int a, int b]){ // Opcional posicionado
  print("a = $a, b = $b");
}

void main(){
  int i = 1;
  int j = 2;
  printar1();
  printar2();
  printar1(a: i, b: j);
  printar2(i, j);
}
```

```
a = null, b = null, c = 3
a = null, b = null
a = 1, b = 2, c = 3
a = 1, b = 2
```

Typedef

```
typedef ManyOperation(int firstNo , int secondNo);

Add(int firstNo,int second){
   print("Add result is ${firstNo+second}");
}
Subtract(int firstNo,int second){
   print("Subtract result is ${firstNo-second}");
}
void main(){
   ManyOperation oper = Add;
   oper(10,20);
   oper = Subtract;
   oper(30,20);
}
```

Typedef é mais usado como uma referência para uma função.

É um tipo de alias para as funções que você quer utilizar.

Pacotes

Caso você importe dois pacotes que contenham identificadores conflitantes, poderá especificar um prefixo para uma ou ambas as bibliotecas.

```dart
import 'package:lib1/lib1.dart';
import 'package:lib2/lib2.dart' as lib2; // NAMESPACE

// Uso do Elemento da lib1.
Element element1 = Element();

// Uso do Elemento da lib2.
lib2.Element element2 = lib2.Element();
```

Importação específica

Caso queira importar apenas uma parte da biblioteca, poderá selecionar os nomes que deseja selecionar.

```dart
// Importando apenas carro.
import 'package:lib1/lib1.dart' show carro;

// Importando todos os nomes exceto carro.
import 'package:lib2/lib2.dart' hide carro;
```

Classes

```dart
class Student {
   String name;
   String get studName {
      return name;
   }
   void set studName(String name) {
      this.name = name;
   }
   Student(String name){
     this.name = name;
   }
   void myName(){
     print(this.name);
   }
}
```

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

Construtores

```dart
class Student {
  String name;

  /*Student(String name){
    this.name = name;
  }*/

  Student(this.name);

  Student.noName(){
    this.name = "NO NAME";
  }

  String toString(){
    return "${this.name}";
  }
}
```

```dart
void main(){
  Student std1 = new Student("Jose");
  Student std2 = Student("Carlos");
  Student std3 = Student.noName();

  print(std1); // Jose
  print(std2); // Carlos
  print(std3); // NO NAME
}
```

POLIMORFISMO

Sistema de Tipos

Como vimos acima Dart é fortemente tipada, portanto ela faz uma inferência de tipos e uma forte verificação de tipos durante a compilação e execução. Entretanto é possível usar um tipo genérico, conhecido como abstract.

Coerção

```
int w = 5;
double x = 5;
x = x + w; // x = x + intToDouble(w);
```

Sobrecarga

Não implementa sobrecarga de métodos, apenas de operadores.

```
void main() {
  print('AAA' + 'BBB');
  print(5 + 5);
}
```

Paramétrico

```
void imprime (dynamic x) {
  print(x);
}

void main() {
  var v;
  int x = 10;

  imprime(x);
  imprime(v);
}
```

Sobrescrita

```dart
class Car {
  void MostrarCarro() => print("CARRO!");
}

class Fusca extends Car {
  @override
  void MostrarCarro() {
    print("FUSCA");
  }
}
```

Inclusão

```dart
class Car {
  void MostrarCarro() => print("CARRO!");
}

class Fusca extends Car {
  @override
  void MostrarCarro() {
    print("FUSCA");
  }
}
```

Classes

```dart
class Student {
   String name;
   String get studName {
      return name;
   }
   void set studName(String name) {
      this.name = name;
   }
   Student(String name){
     this.name = name;
   }
   void myName(){
     print(this.name);
   }
}
```

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

Amarração Tardia

```dart
abstract class Pessoa{
  void falar();  //METODO ABSTRATO
}
class Vitor extends Pessoa{
  @override
  void falar() {
    print("Bom trabalho!");
  }
}
```

Métodos Abstratos

```dart
abstract class Pessoa{
  void falar();  //METODO ABSTRATO
}
class Vitor extends Pessoa{
  @override
  void falar() {
    print("Bom trabalho!");
  }
}
```

EXCEÇÕES

Exceções

- Dart pode lançar e capturar exceções
- Ao contrário de Java todas as exceções são não verificadas
- Os métodos não declaram quais exceções eles podem lançar e você não é obrigado a capturar nenhuma exceção.
- Fornece tipos de exceção e erro, além de vários subtipos predefinidos. É possível definir suas próprias exceções.

Throw

- Um exemplo de lançamento de exceção:

```
throw FormatException('Expected at least 1 section');
```

- É possível lançar objetos arbitrários

```
throw 'Out of llamas!';
```

- Como uma exceção é uma expressão, então é possível lançar exceções em qualquer lugar que permita expressões:

```
void distanceTo(Point other) => throw UnimplementedError();
```

Catch

- A captura da exceção interrompe a sua propagação

```
try {
 teste();
} on ProblemaNoTesteException {
 fazOutroTeste();
}
```

Catch

- Para a captura de mais de uma cláusula de exceção, podemos usar a seguinte estrutura:

```
try {
 teste();
} on ProblemaNoTesteException {
 // Pega uma exceção especifica
} on Exception catch (e) {
 // Pega qualquer outra exceção
} catch (e) {
 // Sem tipo especifico, pega todas
}
```

Catch

- Como é possível observar no exemplo anterior, podemos usar tanto ‘on’ ou ‘catch’ para a cláusula da exceção, use on somente para capturar e catch para utilizar o objeto:

```
try {
 // ...
} on Exception catch (e) {
 print('Detalhes da exceção:\n $e');
} catch (e) {
 print('Detalhes da exceção:\n $e');
}
```

RETHROW

- Para manipular parte da exceção enquanto é propagada, use o ‘rethrow’

```
void main() {
 try {
   try {
     throw 1;
   } catch (e) {
     print("$e");
     rethrow;
   }
 } catch (e2) {
   print("$e2");
 }
}
```

Finally

- Para garantir que algum código seja executado, independentemente de uma exceção ser lançada, use uma cláusula finally.

```
try {
 teste();
} finally {
 // faz o testeFinal
 testeFinal();
}
```

Finally

- É executada após qualquer cláusula de captura correspondente:

```
try {
 procriarMaisLlamas();
} catch (e) {
 print('Erro: $e'); // Manipula a
exceção primeiro
} finally {
 limparBaiaLlamas(); // Para limpar
}
```

CONCORRÊNCIA

Motivação

- O sucesso do Flutter serviu para que desenvolvedores em geral começassem a pensar em dar uma chance para o Dart.
- Suas features desenvolvidas para lidar com uma maneira muito simples com aspectos complicados como concorrência e transições em interfaces serviram para mostrar os valores da linguagem.
- Programação de thread único

ISOLATE

- O Dart usa o Isolates como uma ferramenta para realizar trabalhos em paralelo.
- O pacote `'dart:isolate'` é a solução para obter código de thread único e permitir que o aplicativo faça um uso maior do hardware disponível.
- Sem memória compartilhada, comunicação por mensagens

```
import 'dart:isolate';
```

ISOLATE - EXEMPLO

```dart
import 'dart:isolate';
void foo(var message){
  print('execution from foo ... the message is :${message}');
}
void main(){
  Isolate.spawn(foo,'Hello!!');
  Isolate.spawn(foo,'Greetings!!');
  Isolate.spawn(foo,'Welcome!!');

  print('execution from main1');
  print('execution from main2');
  print('execution from main3');
}
```

- Vamos a um exemplo para entender melhor esse conceito.

Operações assíncronas

- Uma operação assíncrona é executada em um encadeamento, separado do encadeamento principal do aplicativo.

- Palavras chaves *async* - *await*.
- Fazer um trabalho computacional complexo de forma assíncrona é importante para garantir a capacidade de resposta dos aplicativos.

Future

- Semelhante ao *Promise* do JavaScript
- O *Future* é um mecanismo para recuperar o valor de uma tarefa assíncrona após a conclusão, enquanto o Isolates é uma ferramenta para abstrair o paralelismo e implementá-lo em uma base prática de alto nível.

- 3 Estados
    - Incompleta, Completa com dados, Completa com erro

Future - exemplo

```dart
import "dart:async";
import "dart:io";

void main(){
  File file = new File("contato.txt");
   // retorna um future, essa é uma função async
  Future<String> future = file.readAsString();

  // depois que o arquivo é lido, call back method is invoked
  future.then((data) => print(data));

  // isso é impresso primeiro, mostrando que a leitura é async
  print("Fim da main");
}
```

Stream

- Future retorna somente um objeto já o Stream um conjunto
- Future<int> e Stream<int>

Stream - exemplo

- Future retorna somente um objeto já o Stream um conjunto
- Future<int> e Stream<int>

```dart
import 'dart:async';

main() {
 var data = [1,2,3,4,5]; // dados para exemplo
 var stream = new Stream.fromIterable(data);  // criando uma stream

 stream.listen((value) {        //
   print("Received: $value");   //
 });                            //
}
```

- **Saída**

```
Received: 1
Received: 2
Received: 3
Received: 4
Received: 5
```

AVALIAÇÃO DE LPS

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	Sim	Não	Não	Sim
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	**Parcial**	**Sim**	**Sim**	**Parcial**
Confiabilidade	Sim	Não	Não	Sim
Aprendizado				
Eficiência				
Portabilidade	Sim	Não	Não	Sim
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

O programador não tem controle direto do hardware

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	**Sim**	**Não**	**Não**	**Sim**
Aprendizado	Sim	Não	Não	Não
Eficiência				ial
Portabilidade				n
Método de projeto	M			
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Tem Coletor de Lixo.
Faz verificação de índices de vetores.
Não tem aritmética de ponteiros.
Não tem goto.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	**Sim**	**Não**	**Não**	**Não**
Eficiência				
Portabilidade				
Método de projeto				
Evolutibilidade				
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Dart pad, multiparadigma, o sistema cuida do gerenciamento de memória, não possui ponteiros.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	**Parcial**	**Sim**	**Sim**	**Parcial**
Portabilidade	Sim	Não	Não	Sim
Método de projeto	M			
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Possui coletor de lixo.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	**Sim**	**Não**	**Não**	**Sim**
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Possui Máquina Virtual (Dart VM).

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	Sim	Não	Não	Sim
Método de projeto	**Multiparadigma**	**Estruturado**	**Estruturado e OO**	**OO**
Evolutibilidade				
Reusabilidade				
Integração	Sim	Sim	Sim	Parcial
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Orientada a Objeto, Funcional, Imperativa.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	Sim	Não	Não	Sim
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	**Sim**	**Não**	**Parcial**	**Sim**
Reusabilidade				
Integração				
Custo	Baixo	Depende da ferramenta	Depende da ferramenta	Depende da ferramenta

Código simples, modularizado e sem acesso a recursos de baixo nível.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	Sim	Não	Não	Sim
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	**Sim**	**Parcial**	**Sim**	**Sim**
Integração	Sim	Sim	Sim	Parcial
Custo				e da ferramenta

Possui mecanismos de classes e pacotes.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	Sim	Não	Não	Sim
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	**Sim**	**Sim**	**Sim**	**Parcial**
Custo		Depende da	Depende da	Depende da enta

Pode ser compilada para JavaScript.

Critérios Gerais	Dart	C	C++	Java
Aplicabilidade	Parcial	Sim	Sim	Parcial
Confiabilidade	Sim	Não	Não	Sim
Aprendizado	Sim	Não	Não	Não
Eficiência	Parcial	Sim	Sim	Parcial
Portabilidade	Sim	Não	Não	Sim
Método de projeto	Multiparadigma	Estruturado	Estruturado e OO	OO
Evolutibilidade	Sim	Não	Parcial	Sim
Reusabilidade	Sim	Parcial	Sim	Sim
Integração	Sim	Sim	Sim	Parcial
Custo	**Baixo**	**Depende da ferramenta**	**Depende da ferramenta**	**Depende da ferramenta**

A principal framework(flutter) é gratuita e a linguagem é Open Source.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Parcial	Sim	Sim	Sim
Gerenciamento de memória	Sistema	Programador	Programador	Sistema
Persistência dos dados	Biblioteca de classes e funções	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização
Passagem de parâmetros	Por valor e defaut	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável por valor e por cópia de referência.
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Critérios Específicos	Dart	C	C++	Java
Escopo	**Sim**	**Sim**	**Sim**	**Sim**
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos				
Gerenciamento de memória	Sistema	Programador	Programador	Sistema
Persistência dos dados	Biblioteca de classes e funções	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização
Passagem de parâmetros	Por valor e defaut	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável por valor e por cópia de referência.
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Requer definição explícita de entidades.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	**Sim**	**Sim**	**Sim**	**Sim**
Tipos primitivos e compostos				
Gerenciamento de memória	Sistema	Programador	Programador	Sistema
Persistência dos dados	Biblioteca de classes e funções	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização
Passagem de parâmetros	Por valor e defaut	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável por valor e por cópia de referência.
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Ampla variedade de comandos.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	**Parcial**	**Sim**	**Sim**	**Sim**
Gerenciamento de memória				
Persistência dos dados	Biblioteca de classes e funções	Biblioteca de funções	Biblioteca de classes e funções	biblioteca de classes, serialização
Passagem de parâmetros	Por valor e defaut	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável por valor e por cópia de referência.
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Apenas tipos Compostos. Todos os tipos são classes.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Parcial	Sim	Sim	Sim
Gerenciamento de memória	**Sistema**	**Programador**	**Programador**	**Sistema**
Persistência dos dados	B funções	funções	funções	JDBC de classes, serialização
Passagem de parâmetros	Por valor e defaut	Lista variável e por valor	Lista variável, default, por valor e por referência	Lista variável por valor e por cópia de referência.
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Faz uso do Coletor de Lixo.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Parcial	Sim	Sim	Sim
Gerenciamento de memória	Sistema	Programador	Programador	Sistema
Persistência dos dados	**Biblioteca de classes e funções**	**Biblioteca de funções**	**Biblioteca de classes e funções**	**JDBC, biblioteca de classes, serialização**
Passagem de parâmetros	valor	por valor	valor e por referência	cópia de referência.
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Tem funções de I/O. Mas não implementa banco de dados Nativo.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Parcial	Sim	Sim	Sim
Gerenciamento de memória	Sistema	Programador	Programador	Sistema
Persistência dos dados	B			JDBC, de serialização
Passagem de parâmetros	**Por valor e defaut**	**Lista variável e por valor**	**Lista variável, default, por valor e por referência**	**Lista variável por valor e por cópia de referência.**
Encapsulamento e proteção	Sim	Parcial	Sim	Sim

Não existe passagem por referência na versão atual.

Critérios Específicos	Dart	C	C++	Java
Escopo	Sim	Sim	Sim	Sim
Expressões e comandos	Sim	Sim	Sim	Sim
Tipos primitivos e compostos	Parcial	Sim	Sim	Sim
Gerenciamento de memória	Sistema	Programador	Programador	Sistema
Persistência dos dados	Biblioteca de classes e funções	Biblioteca de funções	Biblioteca de classes e funções	JDBC, biblioteca de classes, serialização
Passagem de parâmetros			referência	referência.
Encapsulamento e proteção	**Sim**	**Parcial**	**Sim**	**Sim**

Pacotes e Classes.

Critérios Específicos	Dart	C	C++	Java
Sistema de tipos	Sim	Não	Parcial	Sim
Verificação de tipos	Sim	Estática	Estática / Dinâmica	Estática / Dinâmica
Polimorfismo	Todos	Coerção e Sobrecarga	Todos	Todos
Exceções	Sim	Não	Parcial	Sim
Concorrência	Sim	Não(biblioteca de funções)	Não(biblioteca de funções	Sim

Critérios Específicos	Dart	C	C++	Java
Sistema de tipos	**Sim**	**Não**	**Parcial**	**Sim**
Verificação de tipos				/
Polimorfismo	Todos	Coerção e Sobrecarga	Todos	Todos
Exceções	Sim	Não	Parcial	Sim
Concorrência	Sim	Não(biblioteca de funções)	Não(biblioteca de funções	Sim

Numbers, Strings, Booleans, Lists, Sets, Maps, Runes, Symbols.

Critérios Específicos	Dart	C	C++	Java
Sistema de tipos	Sim	Não	Parcial	Sim
Verificação de tipos	**Estática / Dinâmica**	**Estática**	**Estática / Dinâmica**	**Estática / Dinâmica**
Polimorfismo				
Exceções	Sim	Não	Parcial	Sim
Concorrência	Sim	Não(biblioteca de funções)	Não(biblioteca de funções	Sim

Possui amarração tardia e verificação de índice de vetor.

Critérios Específicos	Dart	C	C++	Java
Sistema de tipos	Sim	Não	Parcial	Sim
Verificação de tipos	Sim	Estática	Estática / Dinâmica	Estática / Dinâmica
Polimorfismo	**Todos**	**Coerção e Sobrecarga**	**Todos**	**Todos**
Exceções				
Concorrência	Sim	de funções)	de funções	Sim

Dart não implementa apenas sobrecarga de métodos.

Critérios Específicos	Dart	C	C++	Java
Sistema de tipos	Sim	Não	Parcial	Sim
Verificação de tipos	Sim	Estática	Estática / Dinâmica	Estática / Dinâmica
Polimorfismo	Todos	Coerção e Sobrecarga	Todos	Todos
Exceções	**Sim**	**Não**	**Parcial**	**Sim**
Concorrência				

Oferece um tratamento amplo de exceções, mas o uso não é obrigatório.

Critérios Específicos	Dart	C	C++	Java
Sistema de tipos	Sim	Não	Parcial	Sim
Verificação de tipos	Sim	Estática	Estática / Dinâmica	Estática / Dinâmica
Polimorfismo	Todos	Coerção e Sobrecarga	Todos	Todos
Exceções	Sim	Não	Parcial	Sim
Concorrência	**Sim**	**Não(biblioteca de funções)**	**Não(biblioteca de funções**	**Sim**

Implementa Async, Future, Stream e Threads.

DART

Daniel Duque
Gustavo Alochio
Lorenzo Moulin
Usiel Lopes