Les bases de Dart

# Présentation

Liens

<https://dart.dev/guides/language/language-tour>

<https://dart.dev/guides/language/effective-dart/style>

## Les bases du langage

Pour le nommage nous utiliserons le [camelCase](https://angular.io/guide/glossary#case-types) (appelé aussi lowerCamelCase).

Les noms de fichiers, librairies, packages, répertoire doivent être en lowercase\_with\_underscores.

Exemple :

library peg\_parser.source\_scanner;

import 'file\_system.dart';

import 'slider\_menu.dart';

### Les types de base

Important : tout est objet donc un type dit primitif comme un int est un objet.

Le mot clé “var” peut être utilisé à la place des types primitifs (Dart le recommande).

**Important** : une variable doit être initialisée. Dart est **orienté “non nullable**”. Mais une variable peut avoir une valeur nulle ssi elle a le mot clé “?”.

Exemple:

String s = 'abc';

~~String s2 =~~ **~~null~~**~~;~~ *// Erreur !!!*

String? s3 = **null**; *// Valide, s3 peut être null*

String value3a = s3; *// Erreur !*

String value3b = (s3==**null** ? **''** : s3); *// OK*

String value3c = s3 ?? **''**; *// OK*

String? s4 = **'texte'**;

String value4 = s4; *// OK*

String? s5 = uneFonctionRetournantDuTexte();

String value5 = s4!; *// Attention si s4 est null ça plante !*

int a = 3;

var b = 2; *// le type est déduit de la valeur*

*// cela s’appelle de l'inférence de type*

const PI = 3.14;

var phrase = 'Ceci est';

phrase = '${phrase} aussi du texte; PI= ${PI}';

### Les listes

List<String> list = []; *// Liste vide*

List<String> list2 = ['one', 'two'];

var list3 = [1, 2, 3];

### Les structures

Les structures classiques : if, switch, while, for

Exemple :

for (String s in list2) {

print(s);

}

### Les fonctions

L’écriture de fonction est classique :

int fonctionA() {

return 1;

}

bool fonctionB(String s1) {

var s2 = s1.toUpperCase();

return s2.startsWith('ABC');

}

### Les énumérations

Le type [enum](https://dart.dev/guides/language/language-tour#enumerated-types) permet de fixer un nombre de valeur (restreint), exemple :

enum Color { red, green, blue }

var color = Color.blue;

if (color == Color.red) {

*// Do something*

}

# 

# Les classes

Nous pouvons utiliser le mot clé “class” comme dans tous les langages orientés objets.

Un membre de classe n’a pas d’étiquette d’accès. Un membre commençant par un underscore est privé sinon il est public.

**Important** : un membre de classe doit être initialisée (comme les variables). Mais un membre peut avoir la valeur nulle ssi il a le mot clé “?”.

Exemple:

class A {

String value1 = 'abc';

String? value2 = null;

String value3; // Erreur

}

class B {

String value; // ok le constructeur fait l'initialisation

B(String val) : value = val;

}

Prenons l’exemple d’une voiture. Une voiture possède les caractéristiques suivantes :

* immatriculation
* nombre de porte (3 ou 5)
* catégorie: familiale, break, SUV
* et bien d'autres …

Ainsi que :

* un constructeur
* des accesseurs
* des méthodes

## Le constructeur

Il ne peut y avoir qu’un seul constructeur du même nom.

### Première version (non recommandée)

En s’inspirant d’un langage objet, la classe Voiture pourrait s’écrire ainsi :

class Voiture {

String immatriculation = '';

int nombreDePortes = 0;

String categorie = '';

Voiture(String immatriculation, int nombreDePortes, String categorie) :

this.immatriculation = immatriculation,

this.nombreDePortes = nombreDePortes,

this.categorie = categorie;

}

Ce n’est pas le style recommandé par Dart.

### Deuxième version (mieux mais ... non)

On prend la première version mais avec le style Dart :

class Voiture {

String immatriculation;

int nombreDePortes;

String categorie;

Voiture(String immatriculation, int nombreDePortes, String categorie) :

this.immatriculation = immatriculation,

this.nombreDePortes = nombreDePortes,

this.categorie = categorie;

}

### Troisième version

On prend la deuxième version mais avec la recommandation Dart :

class Voiture {

String immatriculation;

int nombreDePortes;

String categorie;

Voiture(this.immatriculation, this.nombreDePortes, this.categorie);

}

Le problème ici est que les attributs sont publics … pas très POO mais souvent utilisé en Dart.

### Quatrième version

La même que la troisième version mais avec les membres privés (rappel : un underscore indique un membre privé) :

class Voiture {

String \_immatriculation;

int \_nombreDePortes;

String \_categorie;

Voiture(this.\_immatriculation, this.\_nombreDePortes, this.\_categorie);

}

## Les différents constructeurs

### Avec paramètres nommés

Il faut utiliser les accolades { }

Exemple :

class A {

final int counter;

final String message;

A({required this.counter, required this.message});

}

A a = A(counter: 1, message: 'test');

### Avec paramètres nommés optionnels

Il faut utiliser les accolades { } et fixer les valeurs.

class A {

final int counter;

final String message;

A({this.counter = 1, this.message = 'test'});

}

A a = A(message: 'another test');

### Avec paramètres optionnels

Il faut utiliser les crochets [ ] et fixer les valeurs.

class A {

final int counter;

final String message;

A([this.counter =1, this.message = 'test']);

}

A a1 = A();

A a2 = A(4);

A a3 = A(4, 'Another test');

A a4 = A('Another test'); *// Erreur*

Remarque : vous ne pouvez pas mélanger les crochets [ ] et les accolades { }.

### Les constructeurs nommés

Une classe peut avoir plusieurs constructeurs à condition de les nommer.

Exemple :

class RegularA {

int a = 0;

RegularA();

**RegularA.fromInt(this.a);**

}

var r = RegularA.fromInt(2);

## L’héritage

L’écriture est identique au langage Java.

Exemple :

class Mere {

int a = 0;

}

class Fille extends Mere {

}

## Les classes abstraites et interfaces

Il n’existe pas de mot clé “interface”, il faut utiliser une classe abstraite. L’écriture est identique au langage Java.

Exemple :

abstract class A {

int a = 0;

void method1();

}

Héritage (et donc surcharge, notez qu’un membre attribut peut être surchargé ) :

class B extends A {

@override

int a = 1;

@override

void method1() {

*// TODO: implement method1*

}

}

## Deux façons d’hériter d’une classe

Définissons une classe exemple :

class RegularA {

int a = 0;

void method1() { print('RegularA.method1'); }

RegularA();

RegularA.fromInt(this.a);

}

### Classique

*// La classe RegularA est vu comme une classe*

class B3 extends RegularA {

}

### Comme une classe abstraite

*// La classe RegularA est vu comme une interface*

class B2 implements RegularA {

@override

int a = 2;

@override

void method1() {

*// TODO: implement method1*

}

}

## Les exceptions

Identique aux autres langages, rappel :

**try** {

*// du code*

} **catch** (e) {

*// Une erreur est survenue*

} **finally** {

*// Exécuté quoiqu'il arrive*

}

Nous pouvons créer des classes d’exception :

**class** VoitureException **implements** Exception {

@override

String **get message** => **"VoitureException déclenchée !"**;

}

var ve = VoitureException() ;

print(ve.message);

puis enchaîner les catch :

**try** {

*// du code*

} **on** VoitureException **catch** (e) {

*// Une erreur de type VoitureException est survenue*

} **catch** (e) {

*// Un autre type d'erreur est survenue*

} **finally** {

*// Exécuté quoi qu'il arrive*

}

# 

# Manipulation de liste

Nous allons définir une classe pour les véhicules :

class Vehicule {

String immatriculation;

String constructeur;

String modele;

String couleur;

double prixHT;

Vehicule({required this.immatriculation, required this.constructeur,

required this.modele, required this.couleur, required this.prixHT});

}

Puis nous allons l’alimenter :

List<Vehicule> vehicules = [

Vehicule(immatriculation: 'AA-111-AA', constructeur: 'Peugeot', modele: '208', couleur: 'Gris', prixHT: 20000),

Vehicule(immatriculation: 'AA-222-AA', constructeur: 'Peugeot', modele: '308', couleur: 'Blanc', prixHT: 32000),

Vehicule(immatriculation: 'AA-333-AA', constructeur: 'Peugeot', modele: '3008', couleur: 'Blanc', prixHT: 40000),

Vehicule(immatriculation: 'CC-111-CC', constructeur: 'Audi', modele: 'A4', couleur: 'Gris', prixHT: 42000),

Vehicule(immatriculation: 'DD-111-DD', constructeur: 'Audi', modele: 'SQ5', couleur: 'Noir', prixHT: 72000),

Vehicule(immatriculation: 'DD-444-DD', constructeur: 'Dacia', modele: 'Logan', couleur: 'Noir', prixHT: 8200),

Vehicule(immatriculation: 'FO-777-RD', constructeur: 'Ford', modele: 'Mustang', couleur: 'Blanc glacier', prixHT: 48000),

];

Et nous allons faire des manipulations de listes (les plus courantes).

### Parcourir la liste

#### L’instruction for classique

Exemple:

for(Vehicule vehicule in vehicules) {

print('Véhicule : ${vehicule.immatriculation}');

};

#### La méthode forEach

Permet d’itérer la liste (équivalent de for). forEach ne renvoie rien.

Exemple:

vehicules.forEach((vehicule) {

print('Véhicule : ${vehicule.immatriculation}');

});

### Rechercher un élément

La méthode indexWhere permet de rechercher un élément dans la liste.

Exemple:

var indexVehiculeFord = vehicules.indexWhere((vehicule) => vehicule.immatriculation == 'FO-777-RD');

### Ajouter un élément

La méthode add permet d’ajouter un élément dans la liste.

Exemple:

var citroen2cv = Vehicule(immatriculation: 'CI-222-CV', constructeur: 'Citroën', modele: '2CV', couleur: 'Blanc', prixHT: 12000);

vehicules.add(citroen2cv);

### Supprimer un élément

La suppression d’un élément se fait en 2 temps :

Exemple:

var index = vehicules.indexWhere((vehicule) => vehicule.immatriculation == 'AA-222-AA');

if (index >= 0) {

vehicules.removeAt(index);

}

### Filtrer les éléments

La méthode where permet de filtrer la liste.

Exemple, filtre d’après le prix :

var vehiculePrixSup40ke = vehicules.where((vehicule) => vehicule.prixHT >= 40000);

### Trier les éléments

La méthode sort permet de trier une liste.

Exemple :

vehicules.sort((a, b) => a.prixHT.compareTo(b.prixHT));

### Parcourir une liste pour compter

La méthode fold permet de compter d’après une liste.

Exemple :

double total = list.fold(0, (sum, element) => (sum as int) + element);

# 

# Créer un premier projet

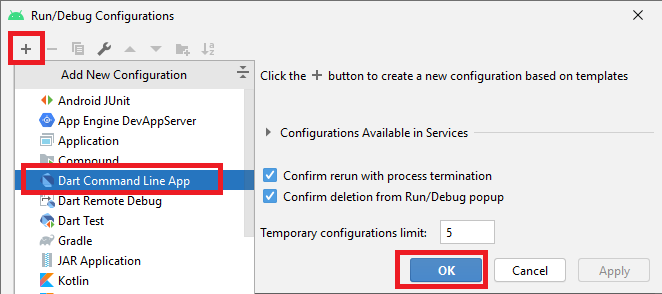
## Création du projet

Placez-vous dans le répertoire de votre choix et ouvrez une console dans ce répertoire.

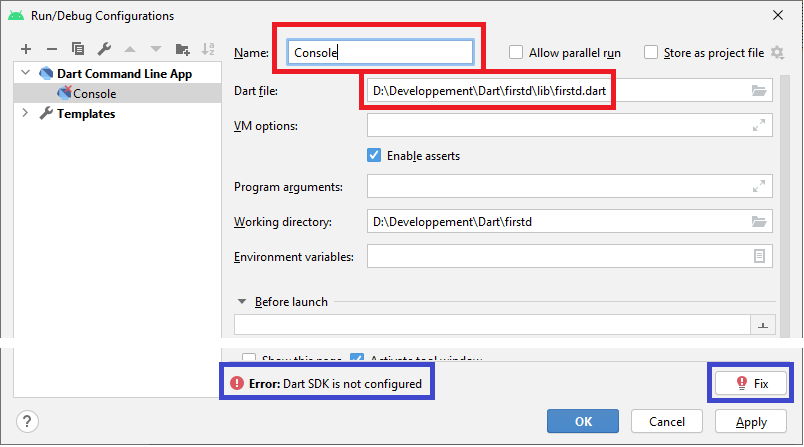
Pour créer le projet, vous devez exécuter la commande suivante : dart create -t console-full firstd.

## Ouverture du projet

Ouvrez le projet avec Android Studio puis configurez-le, cliquez sur le bouton “Add Configuration” : choisissez “Dart Command Line App” et cliquez sur “OK”

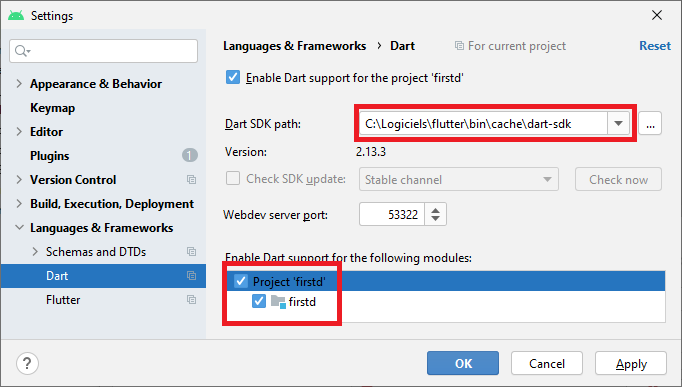


Donnez un nom à la configuration, vérifiez le fichier de démarrage.



Si vous avez l’erreur “Dart SDK is not configured”, cliquez sur le bouton “Fix” puis :

* Sélectionnez le dossier contenant votre SDK
* Sélectionnez le projet courant pour activer Dart



Dans le projet :

* Supprimer le fichier du répertoire lib
* Editer le fichier du répertoire bin (firsttd.dart) et remplacer le code par le suivant :

void main(List<String> arguments) {

print('Hello world !');

}

De même, éditer le fichier du répertoire test (firsttd\_test.dart) et remplacer le code par le suivant :

import 'package:test/test.dart';

void main() {

test('un test', () {

*// plus tard*

});

}

Vous pouvez maintenant exécuter votre projet.