Cours 3. Flutter et Dart

O.R. Merad Boudia

Université d'Oran 1, Ahmed Ben Bella

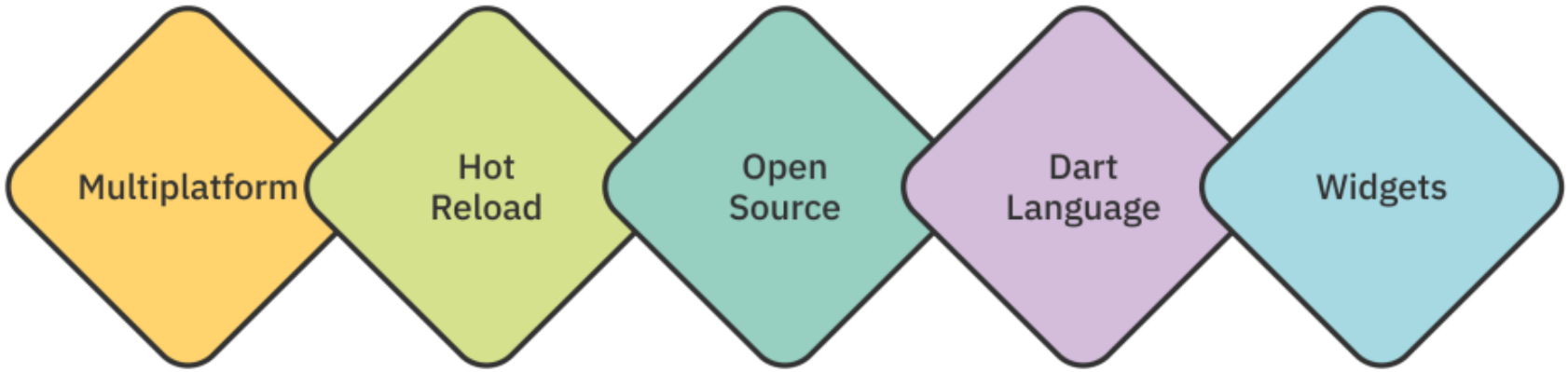
M1 GBM : 2023/2024

Introduction à Flutter

- Flutter est un kit de développement logiciel (SDK) open-source développé par Google.
- Il est utilisé pour créer des interfaces utilisateur (UI) et des applications mobiles multiplateformes.
- Flutter vous permet d'écrire du code une seule fois et de le déployer sur plusieurs plateformes telles qu'Android, iOS et même le web.
- C'est le moyen le plus simple de mettre en place une application et de l'exécuter sur plusieurs plateformes.
- Flutter utilise le langage de programmation **Dart**, également développé par Google.

Introduction à Flutter

- Si vous connaissez Kotlin, Swift, Java ou Typescript, vous trouverez **Dart** familier, car il s'agit d'un langage orienté objet similaire à C.
- Vous pouvez compiler Dart en code natif, ce qui le rend rapide. Il utilise également une machine virtuelle (VM) avec une fonctionnalité spéciale : le rechargement à chaud (**Hot Reload**). Cela vous permet de mettre à jour votre code et de voir les modifications en direct sans le redéployer.



À qui s'adresse Flutter ?

- Flutter est destiné aux développeurs débutants ou expérimentés qui souhaitent démarrer une application mobile avec un minimum de frais généraux.
- Flutter est destiné à quelqu'un qui cherche à créer une application qui s'exécute sur plusieurs appareils.
- C'est pour quelqu'un qui préfère créer des interfaces utilisateur déclaratives avec le soutien d'une grande communauté open source.
- Si vous n'avez pas d'application existante, Flutter est un excellent moyen de développer rapidement quelque chose pour valider une idée ou de créer une application de production complète et multiplateforme.

Les avantages de Flutter ?

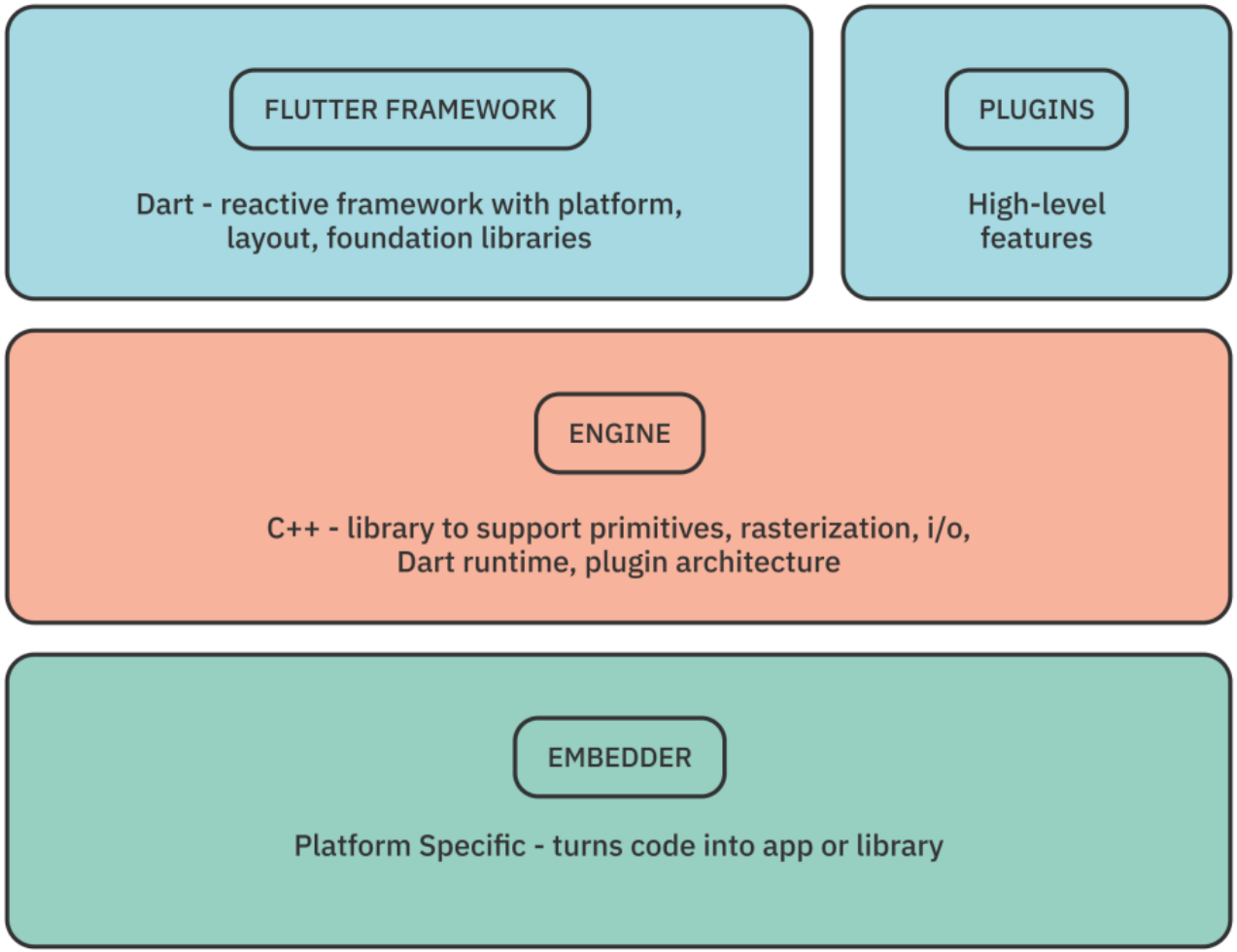
- **Flutter est open-source**. Cela signifie que vous pouvez suivre son évolution et savoir ce qui s'en vient, et même essayer de nouvelles fonctionnalités en cours de développement.
- **Flutter utilise le langage de programmation Dart**. Dart (https://dart.dev) est un langage moderne axé sur l'interface utilisateur qui est compilé en code natif ARM ou x86 ou compilé en Javascript.
- L'une des meilleures fonctionnalités de Flutter est **le rechargement à chaud (Hot Reload)**.
- Flutter est livré avec de **superbes animations et transitions**, et vous pouvez également créer des **widgets personnalisés**.

Quand ne pas utiliser Flutter

- ✓ **Jeux et audio.**
- ➢ Bien que vous puissiez créer des jeux 2D simples à l'aide de Flutter, pour les jeux 2D et 3D complexes, il faudra utiliser une technologie de moteur de jeu multiplateforme comme **Unity** ou **Unreal**.
- ➢ Flutter n'a pas encore de moteur audio sophistiqué, donc les applications d'édition ou de mixage audio sont désavantagées par rapport à celles qui sont spécialement conçues pour une plate-forme spécifique.
- ✓ **Applications avec des besoins spécifiques en matière de SDK natif**
- ➢ Flutter prend en charge de nombreuses fonctionnalités natives, mais pas toutes.

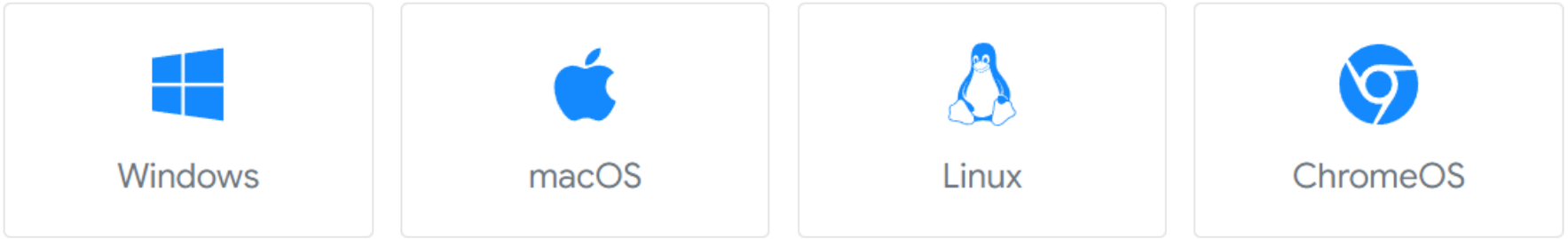
L'architecture de Flutter

- L'architecture Flutter se compose de trois couches principales :

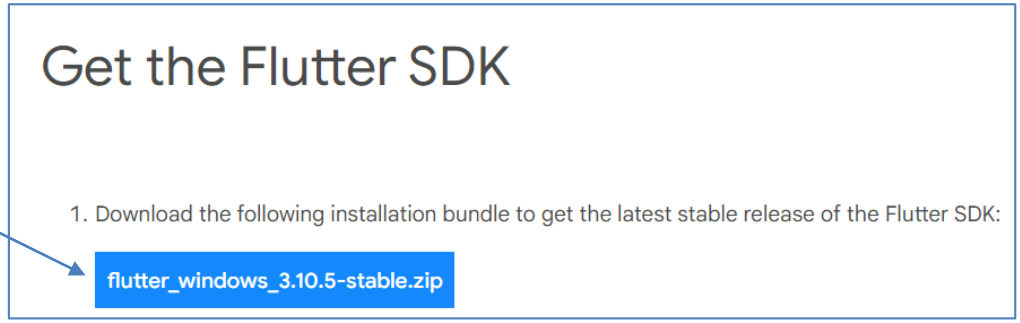


Installation

- ✓ Aller à : https://docs.flutter.dev/get-started/install



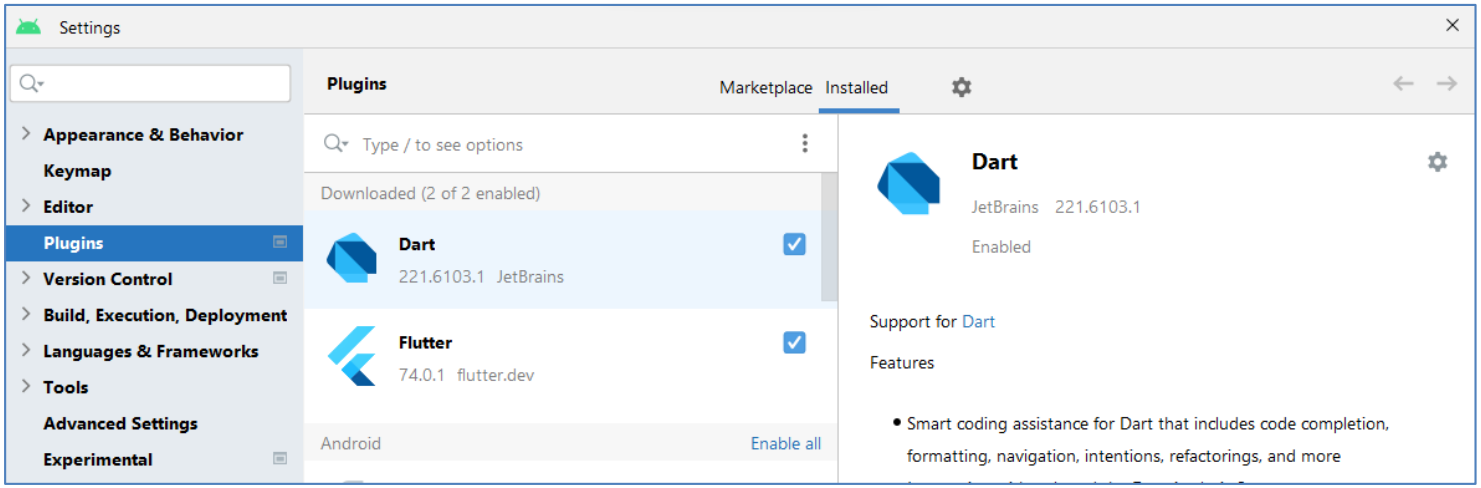
- ✓ Télécharger Flutter SDK :



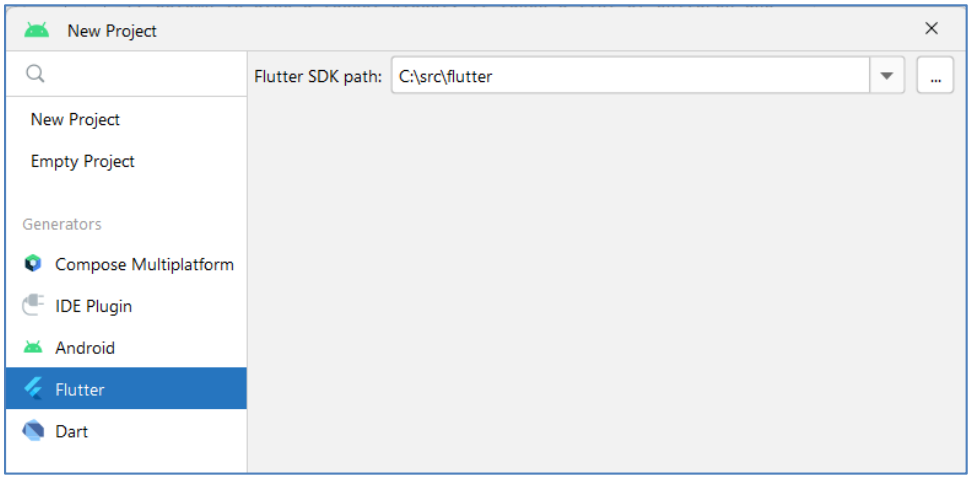
- ✓ Décompresser le ficher et le copier dans : C:\src\flutter
- ✓ Télécharger Android Studio : https://developer.android.com/studio

Installation

✓ Sur Android Studio, allez à Plugin et installez Flutter et Dart :

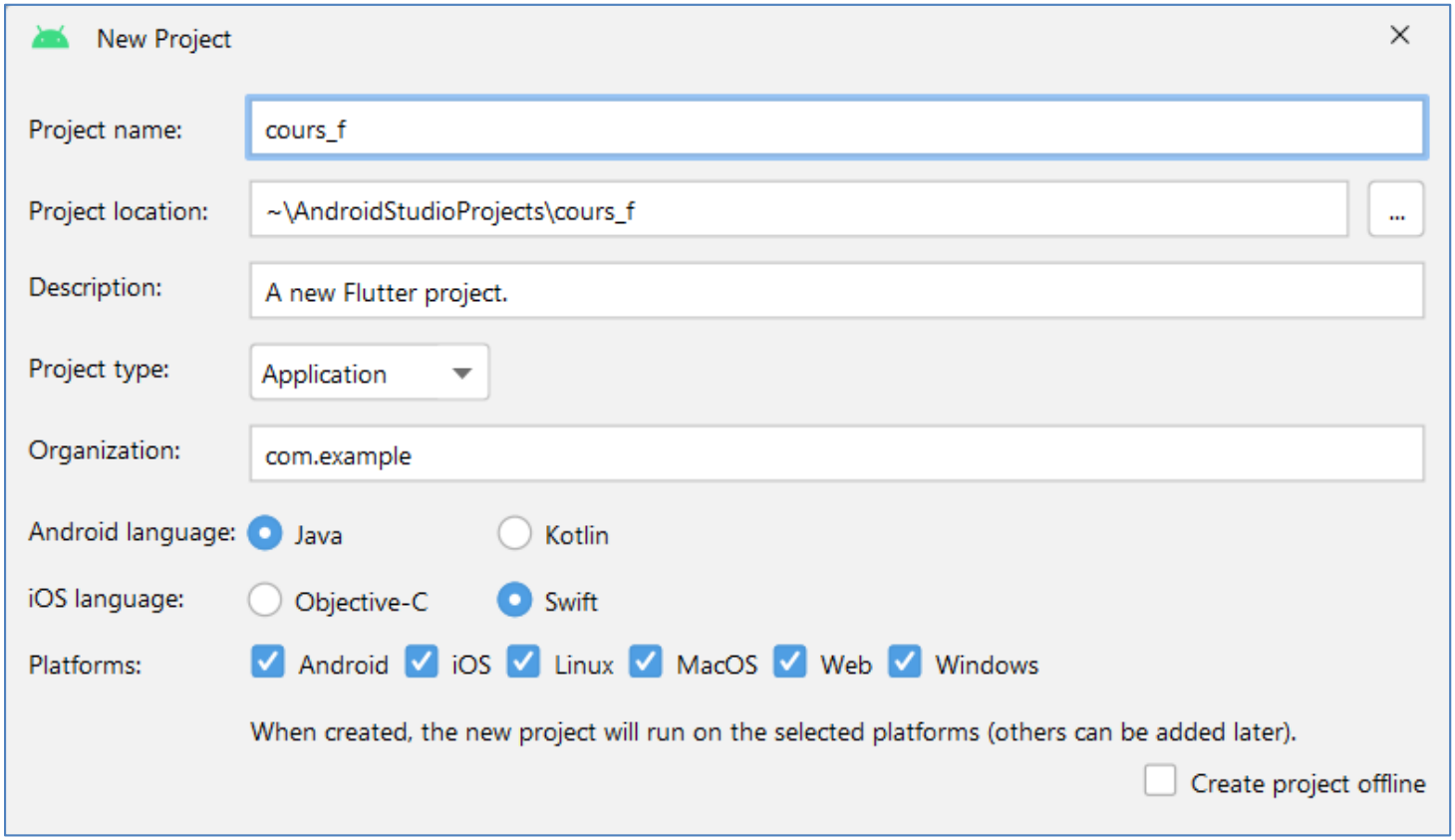


✓ Ajouter le chemin du SDK de Flutter :



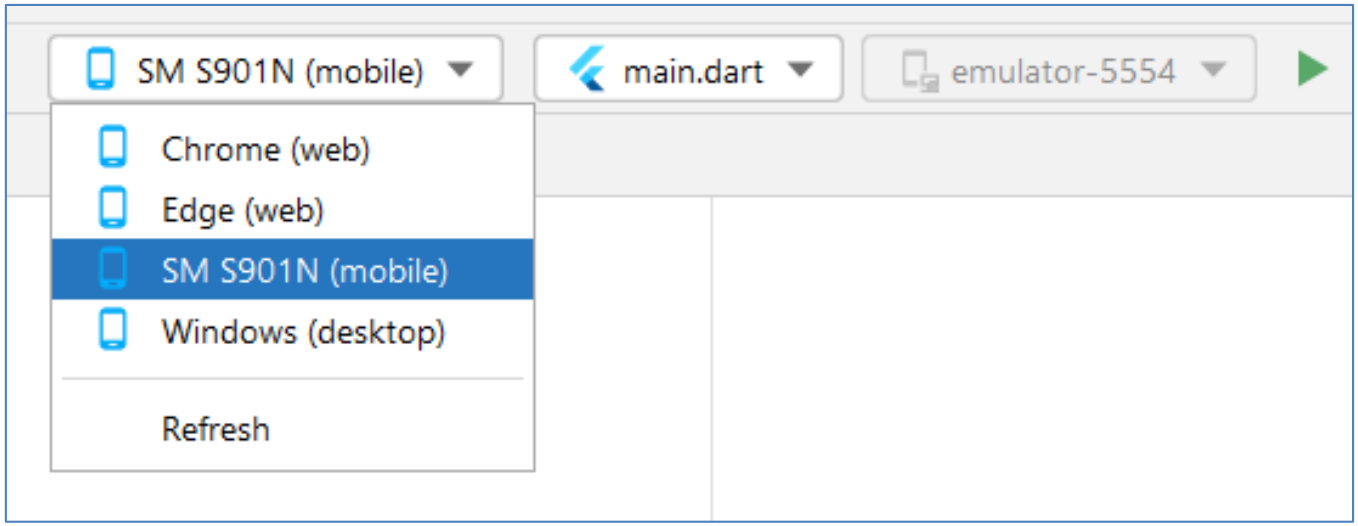
Créer un nouveau projet Flutter

✓ Sur Android Studio, allez à File > New > New Flutter Project > Flutter > Next :



Créer un nouveau projet Flutter

- ✓ Le nouveau projet par défaut est une simple démo Flutter.
- ✓ L'application de démonstration compte le nombre de fois que vous appuyez sur un bouton.
- ✓ Pour exécuter l'application, sélectionnez un navigateur, un appareil connecté, un simulateur iOS ou un émulateur Android.



Le langage Dart

Introduction à Dart

- Nous avons vu que Flutter est un Framework ou un outil dont nous avons besoin pour créer de belles applications mobiles.
- Flutter est écrit en langage de programmation Dart. Pour comprendre le fonctionnement de Flutter, nous devons également comprendre Dart.
- Avant de creuser profondément pour découvrir la relation entre Flutter et Dart, essayons de comprendre un concept clé de la programmation.
- Il y a deux parties distinctes de programmation. L'une est l'abstraction et l'autre la concrétisation. Nous avons besoin de convertir nos idées abstraites en une forme concrète.
- Toute application mobile est une idée abstraite. Nous avons besoin d'un outil comme Flutter pour lui donner une forme concrète.
- Pendant que nous utilisons Flutter, nous rencontrerons de nombreux termes tels que fonction, classe, constructeur, paramètre de position, paramètre nommé, objet, Widget, etc.

Introduction à Dart

- Le langage de programmation Dart est développé par Google.
- Il existe depuis 2011. Cependant, il a récemment gagné en popularité depuis que Google a annoncé le kit de développement logiciel (SDK) Flutter pour développer des applications multiplateformes.
- Dart vise à aider les développeurs à créer efficacement des applications Web et mobiles.
- Dart dispose d'un compilateur **Ahead-of-Time (AOT)** qui compile rapidement du code natif pour la plate-forme cible.
- Il est facile pour les développeurs issus de différents horizons en langage de programmation d'apprendre Dart sans trop d'effort.

La fonction « main » dans Dart

- En Dart, la fonction main est le point d'entrée de tout programme Dart. C'est là que l'exécution du programme commence. La signature de la fonction main en Dart ressemble à ceci :

```dart
void main() {
  // Code de votre programme
}
```

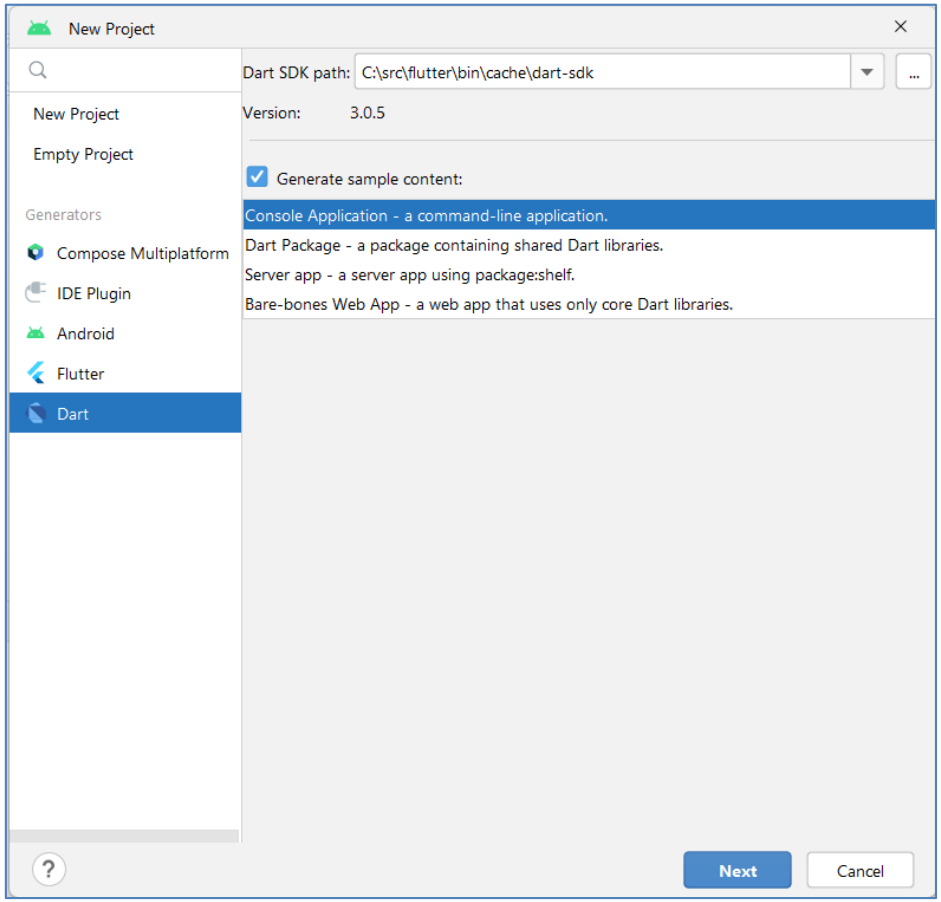
```dart
void main() {
  print("Bonjour, Dart !");
}
```

- ✓ La fonction main a un type de retour **void**, ce qui signifie qu'elle ne renvoie pas de valeur.
- ✓ Vous pouvez ajouter des arguments à la fonction main si vous avez besoin de passer des arguments en ligne de commande à votre programme. Par exemple :

```dart
void main(List<String> arguments) {
  // Code de votre programme
}
```

Créer un projet Dart

- Créez un nouveau projet Dart :



- Nommez le : **cours_dart**

Les fichiers d'un projet Dart

- Lorsque nous créons un projet Dart, il propose un fichier ".dart". Le fichier « cours_dart.dart » se trouve dans le dossier "bin" et un autre fichier du même nom dans le dossier "lib".

bin\cours_dart.dart | lib\cours_dart.dart

```dart
import 'package:cours_dart/cours_dart.dart' as cours_dart;

void main(List<String> arguments) {
  print('Hello world: ${cours_dart.calculate()}!');
}
```

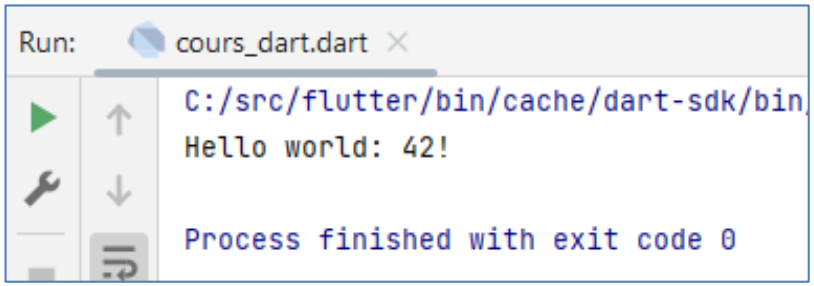
bin\cours_dart.dart | lib\cours_dart.dart

```dart
int calculate() {
  return 6 * 7;
}
```

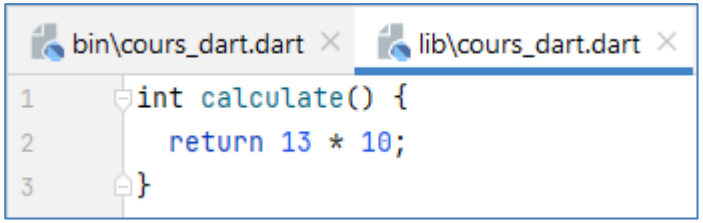
- Comme 'C', 'C++' ou 'Java', l'application Dart passe par la fonction 'main()'. Par conséquent, le fichier 'cours_dart.dart' dans le dossier 'bin' est le fichier principal par lequel notre exemple d'application de console Dart s'exécutera.

Exécution et modification

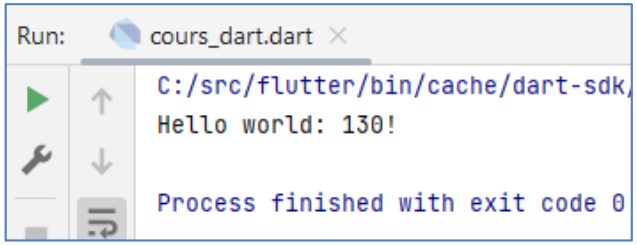
- Si nous exécutons ce code, nous obtenons cette sortie :



- Essayons de modifier les deux fichiers :



- Si nous exécutons ce programme, cela fonctionne et nous donne cette sortie :



Une autre fonction ?

- Pouvons-nous créer une autre fonction dans le dossier "lib" ?
- Laisse-nous essayer. La création d'une nouvelle fonction est un processus très simple.
- Nous allons faire un clic droit sur le dossier 'lib' dans notre Android Studio, il demandera automatiquement de créer différents types de fichiers.
- Nous choisirons Dart avec comme nom « une_nouvelle_fonction". Un nouveau fichier Dart est généré dans le dossier « lib ».
- Nous essayons d'additionner deux nombres via cette fonction et renvoyer le résultat :

Exécution

- Maintenant, nous allons appeler cette fonction à l'intérieur de la fonction main().

bin\cours_dart.dart | lib\cours_dart.dart | une_nouvelle_fonction.dart

```dart
import 'package:cours_dart/cours_dart.dart' as un_objet;
import 'package:cours_dart/une_nouvelle_fonction.dart' as une_nouvelle_fonction;

void main(List<String> arguments) {
  print('Hello world: ${un_objet.calculate()}!');
  print('Additionner 100 et 200: ${une_nouvelle_fonction.additionnerdeuxnombres(100, 200)}');

}
```

- Résultat:

Run: cours_dart.dart

```
C:/src/flutter/bin/cache/dart-sdk/bi
Hello world: 130!
Additionner 100 et 200: 300

Process finished with exit code 0
```

Créer une fonction Void

- Pouvons-nous créer une fonction **void** et l'appeler à l'intérieur d'une fonction principale ? Oui, nous pouvons le faire.

- Ajoutons une fonction **void** dans le fichier 'une_nouvelle_fonction.dart' comme ceci :

bin\cours_dart.dart | lib\cours_dart.dart | une_nouvelle_fonction.dart

```dart

int additionnerdeuxnombres(var x, var y){
return x + y;
}

void neRienFaire(){
print('Ne rien faire');
}
```

- On peut faire appel à cette méthode depuis la méthode main comme ceci:

```dart
void main(List<String> arguments) {
  print('Hello world: ${un_objet.calculate()}!');
  print('Additionner 100 et 200: ${une_nouvelle_fonction.additionnerdeuxnombres(100, 200)}');
  une_nouvelle_fonction.neRienFaire();
}
```

- Résultat:

Run: cours_dart.dart

```
C:/src/flutter/bin/cache/dart-sdk/
Hello world: 130!
Additionner 100 et 200: 300
Ne rien faire

Process finished with exit code 0
```

Introduction à Dart

- Nous pouvons nous demander quelle est la fonction de ces fonctions dans le projet Flutter ?
- Est-ce que ces fonctions auront quelque chose à voir avec notre première application mobile ?
- Comme nous l'avons constaté, lorsque le projet Flutter a été créé, il était accompagné d'un fichier main(), tout comme nous venons de le voir dans le projet Dart.
- L'extrait de code est assez long, mais nous voulons voir le code entier ici, car nous voulons voir si nous pouvons trouver quelque chose de familier en tant que développeur débutant.
- Jusqu'à présent, nous avons appris à créer des fonctions et nous avons entendu dire que tout dans Dart est objet, mais nous ne le comprenons toujours pas très bien.

Dart ---> Flutter

Ce fichier « material.dart » a été fourni par Flutter. Ce fichier possède de nombreuses fonctionnalités de base que nous pouvons appeler.

En regardant le code, nous pouvons dire que, oui, nous avons trouvé une chose familière, une fonction main(). Et la fonction main() ici appelle directement une fonction runApp(); à l'intérieur de cette fonction runApp(), Flutter a passé un paramètre MyApp(), qui ressemble également à une fonction. Mais ce n'est pas une fonction régulière. C'est un objet qui a été instancié à partir de la classe MyApp().

Nous pouvons clairement voir que MaterialApp() appelle une autre fonction ThemeData() à l'intérieur. Par conséquent, nous pouvons appeler une autre fonction à l'intérieur d'une fonction.

dart

```dart
import 'package:flutter/material.dart';

void main() {
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({super.key});
  // This widget is the root of your application
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ),  // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    );  // MaterialApp
  }
}

class MyHomePage extends StatefulWidget {
  const MyHomePage({super.key, required this.title});

  final String title;

  @override
  State<MyHomePage> createState() => _MyHomePageState();
}
```

Dart : Type de données

- Le langage Dart supporte plusieurs types de données qui peuvent être assignés à des variables.

- ✓ Les chaînes de caractères(String)
- ✓ Les nombres (int et double)
- ✓ Les booléens (Vrai ou Faux / True ou False)
- ✓ Les listes (Lists ou Arrays)
- ✓ Les ensembles (Sets)
- ✓ les dictionnaires (Maps)

Dart : Chaînes de caractères

- Une chaîne Dart est une séquence d'unités de code UTF 16. qui représente une séquence de caractères.
- Les valeurs de chaîne dans Dart peuvent être représentées à l'aide de guillemets simples, doubles ou triples.
  - ✓ Les chaînes à une seule ligne sont représentées à l'aide de guillemets simples ou doubles.
  - ✓ Les guillemets triples sont utilisés pour représenter des chaînes multilignes.

Exemple :

```
void main () {
  String s1 = 'chaîne sur une seule ligne';
  String s2 = "chaîne sur une seule ligne";
  String s3 = '''une chaîne
                    de caractères
                    sur plusieurs lignes ''';
  String s4 = """une autre chaîne
                    de caractères
                    sur plusieurs lignes""";

  print(s1);
  print(s2);
  print(s3);
  print(s4);
}
```

```
C:/src/flutter/bin/cache/dart-sdk/bin/dart.
chaîne sur une seule ligne
chaîne sur une seule ligne
une chaîne
                    de caractères
                    sur plusieurs lignes
une autre chaîne
                    de caractères
                    sur plusieurs lignes

Process finished with exit code 0
```

Dart : Chaînes de caractères

- L'opérateur plus (+) est un mécanisme couramment utilisé pour concaténer/interpoler des chaînes.

Exemple :

```
void main() {
  String str1 = "hello";
  String str2 = "world";
  String res = str1+ " " +str2;

  print("La chaîne concaténée est : $res");
}
```

- La propriété de chaîne **isEmpty** renvoie vrai si la chaîne est vide ; sinon renvoie faux.

- La propriété de chaîne **length** renvoie la longueur de la chaîne, y compris les espaces, les tabulations et les sauts de ligne.

Exemple :

```
String str = "Hello All";
print("La longueur de la chaîne est : ${str.length}");
```

Dart : Nombres et Booléens

- Les nombres , dans Dart, peuvent être de 2 types : int et double.

Exemple :

```
int tempNum = 123456;
print(tempNum);

double i = 52.11 ;
print(i);
```

- Double sont essentiellement des valeurs FLOAT plus grandes. Il peut contenir des valeurs décimales fractionnaires.
- Dans Dart, le double prend en charge des valeurs de 64 bits.
- Le type de données booléen est utilisé pour contenir les valeurs vraies et fausses.

Exemple :

```
bool val1 = true;
bool val2 = false;
print(val1);
print (val2);
```

Dart : Exercice 1

❑ **Exercice 1:**

- ✓ Créer un programme Dart qui demande à l'utilisateur de saisir son prénom qui sera stockée dans une variable appelée : **prenom**
- ✓ Sur la ligne suivante, créer une variable String appelée 'votreprenom' et affectez-la à 'Mon nom est **Mohammed**', lorsque l'utilisateur saisi **Mohammed**.
- ✓ Afficher le contenu de la variable 'nom'

Solution :

```dart
import 'dart:io';

void main() {
  print("Bonjour, veuillez saisir votre prénom :");
  String? prenom = stdin.readLineSync();
  String? votreprenom = "Mon prenom est $prenom";
  print(votreprenom);
}
```

Dart : Les variables

- Le mot clé **var** permet de signifier que nous sommes entrain de déclarer une variable.

```dart
var nom = 'Mohammed';
var age = 16;
var isPremium = true;
var langages = ['Python', 'Javascript', 'Java', 'Kotlin'];
```

- Il est strictement interdit de déclarer des variables ayant le même nom. C'est une erreur qui empêchera votre programme d'être exécuté.

- Le mot-clef **dynamic** est également utilisé pour déclarer une variable, ce qui est assez similaire au mot-clef **var**. La différence est que le type de données de cette variable peut être modifié.

```dart
dynamic prenom1 = "omar";
var prenom2 = 16;
prenom1 = 15;
prenom2 = "omar";
```

- Vous pouvez déclarer une variable avec un type de données spécifique au lieu de demander à **Dart** de détecter automatiquement son type de données.

```dart
String myVariable1 = "My Text";
int myVariable2 = 1000;
```

Dart : Exercice 2

- Écrire un algorithme qui calcule la moyenne de trois nombres (A, B et C) et affiche le résultat. Les étapes de l'algorithme sont les suivantes :

- ✓ Lire les valeurs de A, B et C.
- ✓ Calculer la somme de A, B et C.
- ✓ Diviser la somme par 3 pour obtenir la moyenne.
- ✓ Afficher la moyenne calculée.

Pseudo-code

```
1. Lire A
2. Lire B
3. Lire C
4. Somme = A + B + C
5. Moyenne = Somme / 3
6. Afficher "La moyenne est : ", Moyenne
```

- Sous Dart :

Dart

```dart
void main() {
  print("Entrez le premier nombre : ");
  double nombre1 = double.parse(stdin.readLineSync()!);
  print("Entrez le deuxième nombre : ");
  double nombre2 = double.parse(stdin.readLineSync()!);
  print("Entrez le troisième nombre : ");
  double nombre3 = double.parse(stdin.readLineSync()!);

  double moyenne = (nombre1 + nombre2 + nombre3) / 3;
  print("La moyenne des nombres est : $moyenne");
}
```

```
Entrez le premier nombre :
5
Entrez le deuxième nombre :
10
Entrez le troisième nombre :
12
La moyenne des nombres est : 9.0
```

Exécution

Dart : Les constantes

- Pour déclarer une constante, on utilise le mot clé **const** ou **final**.

✓ Souvent, vous savez que vous voudrez une constante dans votre programme, mais vous ne savez pas quelle est sa valeur au moment de la compilation.

✓ Vous ne connaîtrez la valeur qu'après le démarrage du programme. Ce type de constante est appelé constante d'exécution . On utilise dans ce cas **final**.

Exemple :

```
final time = DateTime.now();
print(time);
```

Étant donné que ce code produira des résultats différents selon l'heure de la journée, il s'agit très certainement d'une valeur d'exécution. Donc, pour faire **time** une constante, vous devez utiliser le mot-clé **final** au lieu de **const**.

✓ Une variable **const** doit être constante au moment de la compilation. Une fois qu'une valeur a été assignée à **const**, elle ne peut jamais changer.

```
const time = DateTime.now();
print(time);
```

Const variables must be initialized with a constant value.

Dart : Structures conditionnelle

- Dans Dart, les instructions **if , if-else et if-else-if** sont utilisées pour exécuter un bloc de code basé sur une condition donnée.

- L'expression **switch** sert à contrôler les opérations conditionnelles en mettant en place plusieurs conditions. Elle a le même fonctionnement que la structure conditionnelle **if .. else .. if** tout en offrant une syntaxe plus simple.

```dart
const number = 3;
if (number == 0) {
  print('zero');
} else if (number == 1) {
  print('one');
} else if (number == 2) {
  print('two');
} else if (number == 3) {
  print('three');
} else if (number == 4) {
  print('four');
} else {
  print('something else');
}
```

```dart
const number = 3;
switch (number) {
  case 0:
    print('zero');
    break;
  case 1:
    print('one');
    break;
  case 2:
    print('two');
    break;
  case 3:
    print('three');
    break;
  case 4:
    print('four');
    break;
  default:
    print('something else');
}
```

Dart : Les boucles

- Comme dans la plupart des langages, la boucle **for** dans le langage Dart, permet de répéter une expression un certain nombre de fois.

```
for (var compteur = 1; compteur <= 3 ; compteur++){
  print("la valeur du compteur est $compteur");
}
```

```
C:/src/flutter/bin/cache/dart-
la valeur du compteur est 1
la valeur du compteur est 2
la valeur du compteur est 3
```

- La boucle **For…in** dans Dart prend une expression ou un objet comme itérateur. Il est similaire à celui de Java et son flux d'exécution est également le même que celui de Java.

```
var langues = ['Français', 'Anglais', 'Espagnol', 'Arabe'];
for (var l in langues){
  print(l);
}
```

```
C:/src/flutter/bin/c
Français
Anglais
Espagnol
Arabe
```

- La boucle **do..while** répète une séquence d'instructions tant que la condition est vraie.

```
int i = 4;
int j = 0;
do {
  print("index : $j");
  j += 1;
} while (j <= i);
```

```
C:/src/flutter/bin/cache
index : 0
index : 1
index : 2
index : 3
index : 4
```

Les collections : Les listes

```dart
/// Liste
List ListeVide = [];
List UneListe = ['Dart', 'Java', 'Kotlin'];
/// Set
// Creation d'un ensemble de chaines de caracteres
Set langSet = {'Dart', 'Kotlin', 'Swift'};
Set sdkSet = {'Flutter', 'Android', 'iOS'};

void main() {
  dynamic result;
  print('==== Liste ====');
  //Verifier si la liste est vide
  result = ListeVide.isEmpty;
  print(result);
  // Utiliser map pour transformer les éléments d'une liste
  result = UneListe.map((e) => "$e Language").toList();
  print(result);
  // Filtrer les mots contenant la lettre 'a'
  result = UneListe.where((element) => element.toString().contains('a'));
  print(result);
  // Set
  print('==== Set ====');
  // Ajouter 'Java' to langSet
  langSet.add('Java');
  print(langSet);
  langSet.remove('Java');
  // Ajouter plusieurs elements
  langSet.addAll(['C#', 'Java']);
  sdkSet.addAll(['C#', 'Xamarin']);
  print(langSet);
  print(sdkSet);
  // Trouver les elements communs
  result = langSet.intersection(sdkSet);
  print(result); // C#
  // Trouver l'union de deux ensembles.
  result = langSet.union(sdkSet);
  print(result);
```

```
==== Liste ====
true
[Dart Language, Java Language, Kotlin Language]
(Dart, Java)
==== Set ====
{Dart, Kotlin, Swift, Java}
{Dart, Kotlin, Swift, C#, Java}
{Flutter, Android, iOS, C#, Xamarin}
{C#}
{Dart, Kotlin, Swift, C#, Java, Flutter, Android, iOS, Xamarin}
```

Les collections : Map

```dart
/// Map
// Creation d'une Map de clés int et des valeurs string.
var intToStringMap = Map<int, String>();
// Creation d'une Map de clés string et des valeurs string.
var techMap = {
  'Flutter': 'Dart',
  'Android': 'Java',
  'iOS': 'Swift',
};
void main() {
  dynamic result;
  // Map
  print('==== Map ====');
  intToStringMap[1] = '1';
  intToStringMap[2] = '2';
  intToStringMap[3] = '4';
  // Première entrée de la map
  result = intToStringMap.entries.first;
  print(result);
  // dernière entrée de la map
  result = intToStringMap.entries.last;
  print(result);
  // Retourne un booléen. true si la clé est trouvée, sinon false
  result = techMap.containsKey('Flutter');
  print(result);
  // Verifier si la clé est présente dans la map
  result = techMap.containsValue('Dart');
  print(result);
  // afficher toutes les valeurs
  techMap.values.forEach((element) {
    print("$element");
  });
  // Parcourir et afficher toutes les paires (clé, valeur)
  techMap.entries.forEach((element) {
    print(
        "${element.value} is used for developing ${element.key} applications.");
  });
}
```

```
==== Map ====
MapEntry(1: 1)
MapEntry(3: 4)
true
true
Dart
Java
Swift
Dart is used for developing Flutter applications.
Java is used for developing Android applications.
Swift is used for developing iOS applications.
```

Les fonctions

```dart
// Fonction nommée
bool isFlutter(String str) {
  return str == 'Flutter';
}
// Passer une fonction comme paramètre
int calculate(int value1, int value2, Function(int, int) function) {
  return function(value1, value2);
}

int subtract(int a, int b) {
  return a > b ? a - b : b - a;
}
// Fonction en une ligne
int add(int a, int b) => a + b;
// Fonction anonyme
Function anonymousAdd = (int a, int b) {
  return a + b;
};

void main() {
  dynamic result;
  // Utiliser les fonctions nommées
  result = isFlutter("Flutter");
  print(result);
  //Passer la fonction 'substract' comme paramètre
  result = calculate(5, 4, subtract);
  print(result);
  // Utiliser une fonction en une ligne
  result = add(5, 4);
  print(result);
  // Utiliser les fonctions anonymes
  result = anonymousAdd(4, 5);
  print(result);
}
```

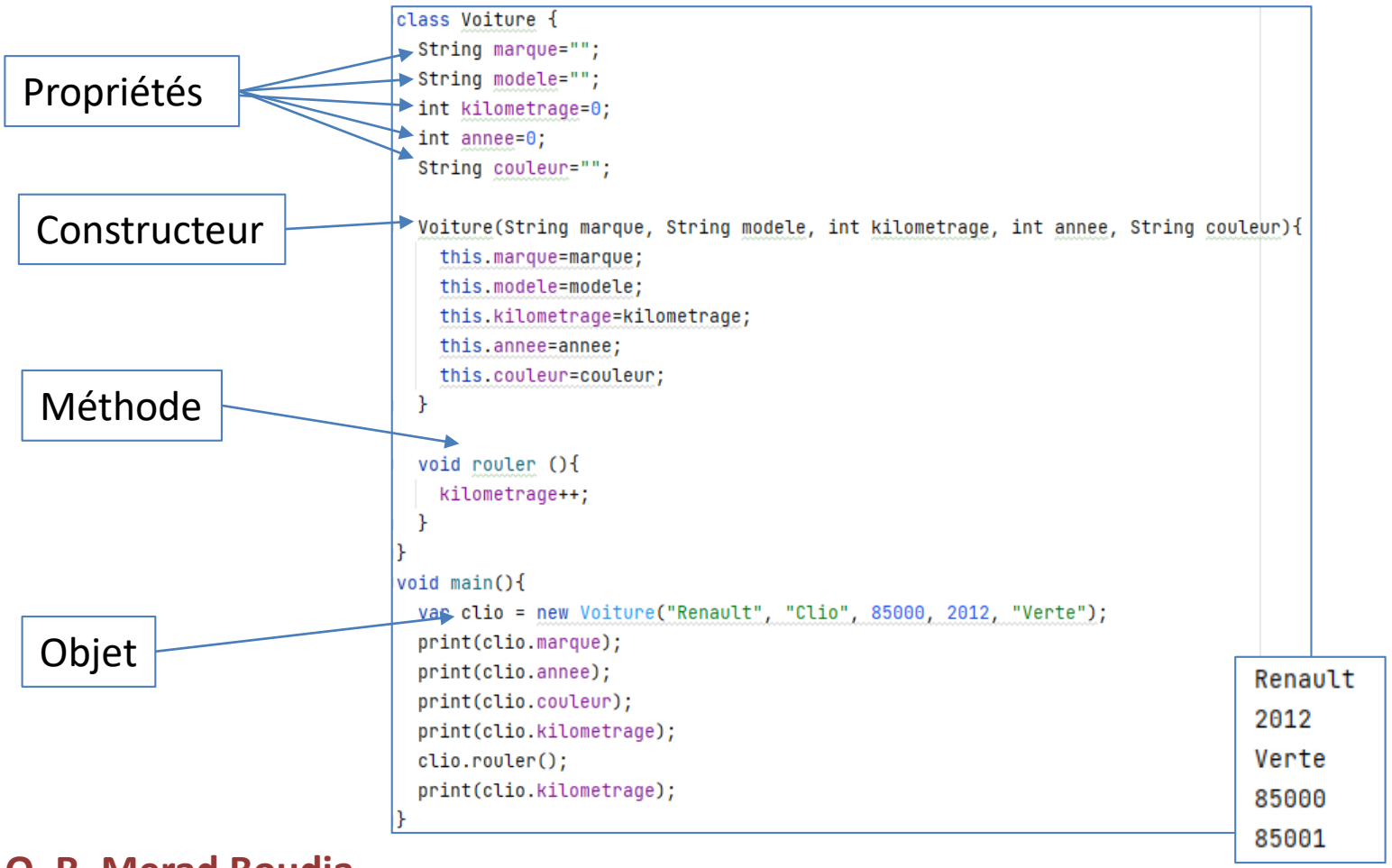
```
C:/src/flutter/bi
true
1
9
9

Process finished
```

Dart
La programmation orienté objet

Dart : POO (Classe et objet)

- Dart est un langage orienté objet avec des classes. Chaque objet est une instance d'une classe, et toutes les classes sauf **Null** descendent de Object.

- Exemple de classe:



Dart : POO (Héritage)

- L'héritage est la pierre angulaire de la programmation objet. En utilisant le mot clé **extends**, vous pouvez permettre à **une classe enfant** d'hériter les propriétés et les méthodes d'une autre classe parente. De plus, la classe enfant a accès aux propriétés et méthodes de sa propre classe ainsi qu'à la classe de base.

- Exemple :

```dart
class Cabriolet extends Voiture{
  int tempsdecapotage = 3;
  void toitOuvrant(){
  print("J'ai un toit ouvrant");
  }
  Cabriolet (String marque, String modele, int kilometrage, int annee, String couleur)
      :super(marque,modele,kilometrage,annee, couleur){

  }
}

void main(){
  var clio = new Voiture("Renault", "Clio", 85000, 2012, "Verte");
  print(clio.marque);
  print(clio.annee);
  print(clio.couleur);
  print(clio.kilometrage);
  clio.rouler();
  print(clio.kilometrage);
  print("##################");
  var cc = new Cabriolet("Peugeot","207cc", 100000, 2012, "Blanche");
  print(cc.marque);
  print(cc.annee);
  print(cc.couleur);
  print(cc.kilometrage);
  cc.toitOuvrant();
  cc.rouler();
  print(cc.kilometrage);
}
```

le mot-clé **super** permet d'initialiser les propriétés de la superclasse avant d'initialiser les spécificités de la sous-classe.

```
Renault
2012
Verte
85000
85001
##################
Peugeot
2012
Blanche
100000
J'ai un toit ouvrant
100001
```

Dart : POO (Héritage)

Classe mère

Classe enfant 1

Classe enfant 2

```dart
class Patient {
  String nom;
  int age;
  Patient(this.nom, this.age);
  void afficherInfo() {
    print("Nom: $nom, Âge: $age");
  }
}
class PatientNormal extends Patient {
  String diagnostic;
  PatientNormal(String nom, int age, this.diagnostic)
      : super(nom, age);
  void afficherDiagnostic() {
    print("Diagnostic: $diagnostic");
  }
}
class PatientUrgent extends Patient {
  String conditionUrgente;
  PatientUrgent(String nom, int age, this.conditionUrgente)
      : super(nom, age);
  void afficherConditionUrgente() {
    print("Condition Urgente: $conditionUrgente");
  }
}

void main() {
  // Créer un patient normal
  PatientNormal patientNormal = PatientNormal("Alice", 30, "Rhume");
  print("Informations du patient normal:");
  patientNormal.afficherInfo();
  patientNormal.afficherDiagnostic();
  // Créer un patient urgent
  PatientUrgent patientUrgent = PatientUrgent("Bob", 45, "Crise cardiaque");
  print("\nInformations du patient urgent:");
  patientUrgent.afficherInfo();
  patientUrgent.afficherConditionUrgente();
}
```

```
Informations du patient normal:
Nom: Alice, Âge: 30
Diagnostic: Rhume

Informations du patient urgent:
Nom: Bob, Âge: 45
Condition Urgente: Crise cardiaque
```

Dart : POO (Le polymorphisme)

- Le polymorphisme:

Classe mère

Méthode de la classe mère

Classe enfant 1

Redéfinition de la méthode

Classe enfant 2

Redéfinition de la méthode

```dart
class Forme{
  int largeur=0;
  int longueur=0;
  Forme(int largeur, int longueur){
    this.largeur=largeur;
    this.longueur=longueur;
  }
  void calculerAire(){
    print ("Calcul de l'aire en cours....");
  }
}
class Triangle extends Forme {
  Triangle(int largeur, int longueur) :super(largeur, longueur) {}
  @override
  void calculerAire(){
    super.calculerAire();
    var aire = largeur*longueur/2;
    print ("Aire = $aire");
  }
}
class Carre extends Forme{
  Carre(int largeur, int longueur):super(largeur,longueur){}
   @override
   void calculerAire(){
      super.calculerAire();
      var aire = largeur*longueur;
      print ("Aire = $aire");
   }
}
void main(){
 var tri=new Triangle(4, 5);
 tri.calculerAire();
 var ca=new Carre(3, 4);
 ca.calculerAire();
}
```

```
Calcul de l'aire en cours....
Aire = 10.0
Calcul de l'aire en cours....
Aire = 12
```

En Dart, le polymorphisme vous permet de substituer des méthodes d'une classe mère par des méthodes de classe enfant. Cela permet d'appeler la méthode appropriée en fonction du type de l'objet.

Dart : POO (Enum)

- Les énumérations:

```dart
enum Flash{
  auto,
  off,
  on
}
void main(){
  var appareil = Flash.on;

  switch (appareil){
    case Flash.auto:
      print("Flash auto");
      break;
    case Flash.off:
      print("Flash off");
      break;
    case Flash.on:
      print("Flash on");
      break;
  }
}
```

```
Flash on
```

Les énumérations sont utilisées pour représenter un ensemble fini de valeurs constantes.

Dart : POO

```dart
class Person {
  String name;
  int age;
  String food;
  // Le constructeur
  Person(this.name, this.age, this.food);
  // Getter
  String get personName => this.name;
  // Setter
  set personName(String value) => this.name = value;
  //Methode
  void eats(String food) {
    this.food = food;
  }
  //Affichage
  String toString() {
    return "Mon nom est $name, et j'aime $food";
  }
}
void main() {
  Person person = Person("Mohammed", 20, "La Pizza");
  print(person.toString());

  Person child = Person("Ali", 15, "La Pizza");
  child.eats("Les Bananes");
  print(child.toString());

  child
    ..name = 'Omar'
    ..eats("Les Pâtes");
  print(child.toString());

  child
    ..personName = 'Reda'
    ..eats("Les Tomates");
  print(child.toString());
}
```

Dart prend en charge les syntaxes en cascade. Il est utile d'attribuer les valeurs aux propriétés et aux méthodes à la fois en utilisant deux points.

```
C:/src/flutter/bin/cache/dart-sdk/bin/dart
Mon nom est Mohammed, et j'aime La Pizza
Mon nom est Ali, et j'aime Les Bananes
Mon nom est Omar, et j'aime Les Pâtes
Mon nom est Reda, et j'aime Les Tomates

Process finished with exit code 0
```

Références

- ✓ Thomas Bailey , Alessandro Biessek, et al. Flutter for Beginners: **Cross-platform mobile development from Hello, World! to app release with Flutter 3.10+ and Dart 3.x**, Kindle, 2023.
- ✓ Sanjib Sinha. Beginning **Flutter with Dart: A Beginner to Pro. Learn how to build Advanced Flutter Apps** (Flutter, Dart and Algorithm Book 1), Kindle Edition, 2021.
- ✓ Mark Clow. **Learn Google Flutter Fast: 65 Example Apps**, Paperback, 2019.