Chapitre 4
Dart, les fondamentaux

1. Introduction

Dart est un langage orienté objet moderne apparu en 2010 à l'initiative de deux développeurs de Google : Lars Bak et Kasper Lund.

À l'origine, ce projet avait comme objectif de remplacer JavaScript à moyen ou long terme, en raison des difficultés à faire évoluer ce dernier. Google a donc conçu Dart à partir de cette idée et la version 1.0 est sortie le 10 octobre 2011.

Le langage a même été adopté par ECMA International en 2014. Pour rappel, il s'agit d'une organisation européenne de standardisation en informatique. Parmi ses responsabilités, elle adopte des standards notamment pour les langages de programmation (y compris de scripts).

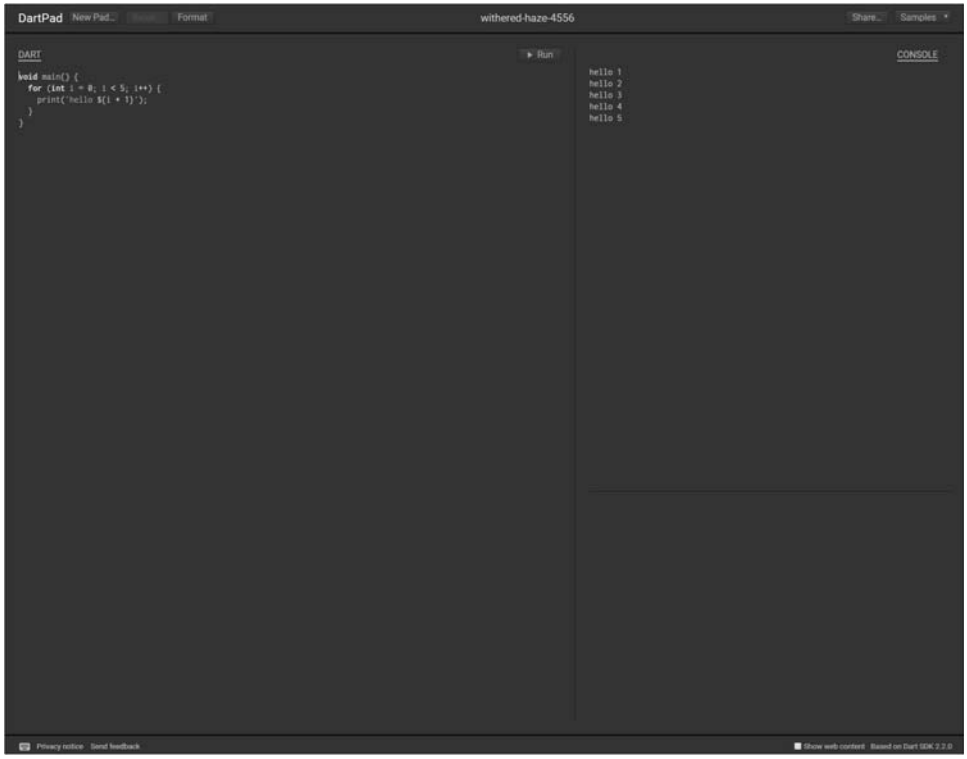
Malgré tous les efforts de Google, Dart n'a jamais percé et JavaScript reste bien établi sur son piédestal. Le langage est donc tombé petit à petit dans l'oubli jusqu'à l'arrivée de Flutter courant 2018.

La volonté de pousser le SDK au-devant de la scène, entre autres, a conduit Google à sortir une nouvelle version plus simple, plus rapide et plus intuitive de Dart en début d'année 2018, la 2.0.

C'est cette dernière version que nous allons découvrir ici, puisque, comme nous avons déjà pu l'expliquer dans le premier chapitre, Flutter s'appuie sur ce langage. Il est primordial d'en posséder les bases avant de pouvoir pleinement profiter de tous les avantages que nous procure Flutter.

Afin de mettre en œuvre et de tester les exemples qui vont suivre, il est bon de noter l'existence d'un outil extrêmement utile : `DartPad`.

Vous pourrez le trouver à l'adresse suivante : https://dartpad.dartlang.org/



DartPad de Google

2. Syntaxe de base

Dans un premier temps, il convient de passer en revue, la syntaxe de base de Dart. Nous allons donc découvrir comment utiliser les variables, les constantes, les collections, les alternatives ainsi que les boucles. Le but du jeu n'est pas d'être exhaustif, mais bien de nous outiller avec les éléments indispensables pour la suite.

2.1 Variables

Dart supporte un certain nombre de types comme les nombres, les chaînes de caractères, les booléens, ou encore des collections comme les listes simples ou les listes fonctionnant avec un couple clé/valeur. Nous aborderons ces deux dernières notions un peu plus loin dans ce chapitre.

2.1.1 Les nombres

Syntaxe de base

Concernant les nombres, on peut noter la possibilité d'utiliser les entiers (Integer) ou les nombres réels (Double).

L'écriture va se faire de manière assez classique :

```
int monEntier;
double monReel;
```

Remarque

À ce moment-là, si la variable n'est pas initialisée avec une valeur choisie, elle prendra `null` comme valeur par défaut. Ce traitement est également valable pour les nombres (entier ou réel) puisqu'en Dart, tout est considéré comme objet. C'est un point notable puisque cela diverge de certains langages.

L'initialisation est réalisée comme suit :

```
int monEntier = 2;
double monReel = 1.856;
```

Héritage de la classe num

Les `int` et les `double` héritent de la classe `num` (les notions d'héritage seront développées plus amplement dans le prochain chapitre). Il est donc possible d'utiliser ce dernier pour garantir que la variable est bien un nombre de type `int` ou `double`. Si ce n'est pas le cas, une erreur de compilation se produira.

```
num unNombre = 318;
num unNombreDifferent = 3.265464;
```

La manipulation des nombres

Il est possible de mener différentes opérations sur les nombres telles que des conversions depuis une chaîne de caractères. Pour cela, il suffit d'utiliser la syntaxe suivante :

```
int.tryParse(valeurAConvertir) ;
```

Par exemple, pour convertir la chaîne de caractère « 32 » en entier et l'affecter immédiatement dans une variable il suffit d'écrire ceci :

```
int nombreDepuisChaineDeCaracteres = int.tryParse("32");
```

2.1.2 Les chaînes de caractères

Syntaxe de base

La déclaration et l'initialisation des chaînes de caractères est semblable à celle des nombres :

```
String maChaine = 'Hello World ! ';
```

La valeur de la chaîne est entourée de simple ou double *quote*.

La manipulation des chaînes de caractères

La conversion

Comme dans tant d'autres langages, il existe beaucoup de possibilités qui sont offertes concernant la manipulation de ces chaînes.

On peut noter d'abord la conversion depuis un nombre grâce à la fonction `toString()`. Sa syntaxe est des plus simple :

```
maChaine.toString() ;
```

Il ne faut pas hésiter à utiliser cette fonction très puissante. D'autant qu'il est possible, comme décrit dans le chapitre suivant, de venir modifier le résultat de cette dernière. Voici un exemple :

```
int conv = 12;
String numConv = conv.toString();
```

La mise en majuscule ou minuscule

Des opérations classiques de mise en minuscules ou majuscules existent également, elles utilisent respectivement les syntaxes suivantes :

`maChaine.toLowerCase()`

ou

`maChaine.toUpperCase()`

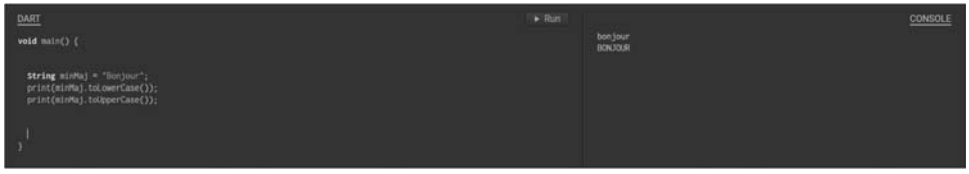
En prenant l'exemple d'une chaîne de caractères nommée `minMaj` qui aurait pour valeur « Bonjour », il serait possible d'utiliser les deux fonctions comme ceci :

```
String minMaj = "Bonjour";
 print(minMaj.toLowerCase());
 print(minMaj.toUpperCase());
```

Remarque

La fonction `print()` sert à afficher la valeur d'un objet dans la console.

Le résultat de ces opérations sera le suivant :



Résultats des fonctions `toLowerCase()` et `toUpperCase()`

Le découpage de la chaîne de caractères

Enfin, il est également possible de découper une chaîne grâce à la fonction `substring()`. La syntaxe à utiliser est la suivante :

```
maChaîne.substring(valeurDeDepartIncluse, ValeurDeFinExcluse) ;
```

Concrètement, il est possible d'avoir l'exemple suivant :

```
print(minMaj.substring(0,3));  //Bon
```

Le premier paramètre est le point de départ de la scission. Dans l'exemple, le 'B' correspond à la position 0 dans la chaîne de caractères. Ce paramètre est donc inclus dans le résultat.

Le second paramètre, 3, est quant à lui le terminus et sera exclu. Dans l'exemple, il s'agit du 'j'.

La console affichera donc 'Bon'.

2.1.3 Les booléens

Un autre type très courant et qui sera extrêmement utile est le type booléen. Prenant les valeurs vrai ou faux, il servira notamment dans l'établissement d'alternatives.

L'écriture se fait comme suit :

```
bool choix1 = true;
bool choix2 = false;
```

2.1.4 Types dynamiques

Le type var

Enfin, il peut être intéressant d'évoquer les types `var` et `dynamic`. Ils sont utilisés quand on ne souhaite pas typer une variable au moment de sa déclaration. Au moment de l'initialisation, le système, en fonction de la valeur passée, déterminera lui-même le type approprié.