Développement d'applications mobiles

Chapitre 2 - Dart - concepts de base

Daniel Schreurs

1er octobre 2022

Haute École de la Province de Liège

Table des matières i

Table des matières ii

Table des matières iii

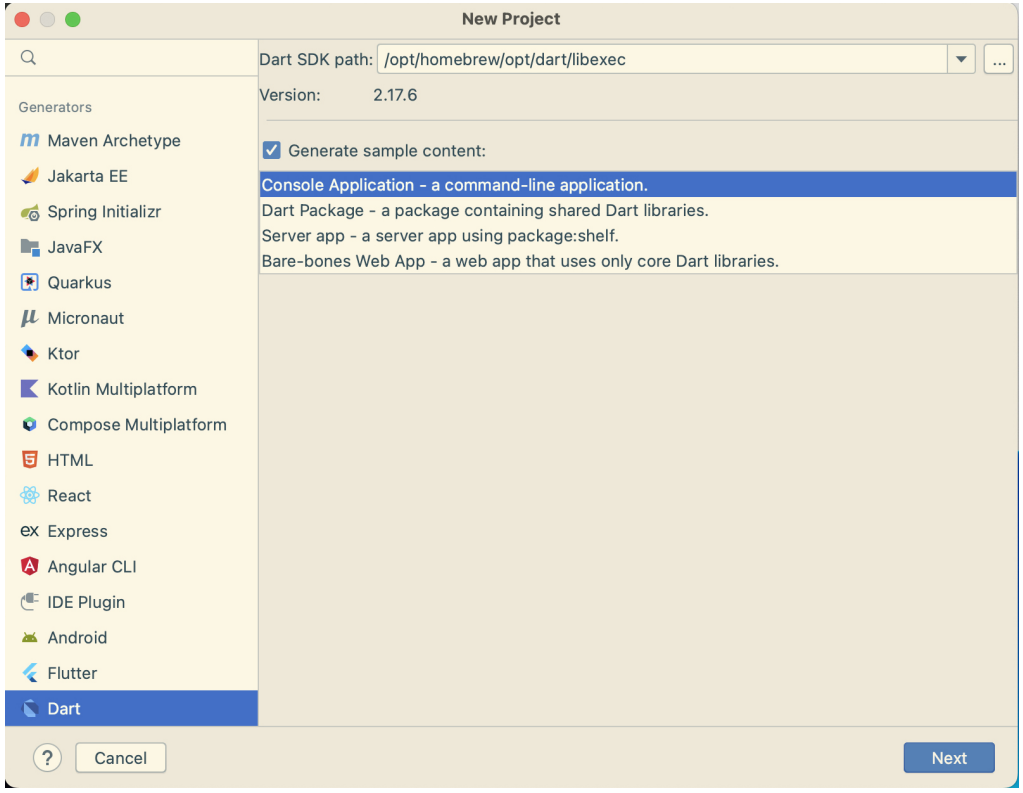
Dart

Dart : Présentation

- Deux développeurs de Google : Lars Bak et Kasper Lund (2010) ;
- Version 1.0 : 2011 ;
- Adopté par ECMA International en 2014 ;
- Version 2.0. 2018.

Dart : DartPad

- Éditeur entièrement en ligne et gratuit ;
- Permet d'isoler une problématique à la fois.



Nouveau projet console - Dart

Variables

Variables : Les nombres

- Les entiers (Integer) ;
- les nombres réels (Double).

```
1        int monEntier;
2        double monReel;
```

Important
Si la variable n'est pas initialisée avec une valeur choisie, elle prendra `null` comme valeur par défaut.

Variables : Les chaines de caractères

- La déclaration et l'initialisation des chaines de caractères sont semblables à celle des nombres ;
- La valeur de la chaine est entourée de simples ou doubles guillemets ;
- Conversion depuis un nombre (ou autre) grâce à la fonction `toString()` ;
- Une chaine de caractères est un objet, on peut appeler d'autres méthodes `toLowerCase()`.

```
1 void main(List<String> arguments) {
2   String v1;
3   v1 = "tout le monde";
4   print("bonjour".toUpperCase());
5   print("BONJOUR".toLowerCase());
6   print("Bonjour $v1");// sans {}
7   print("Bonjour ${v1.toUpperCase()}");// avec {}
8 }
```

Variables : Les booléens

- Un autre type très courant ;
- extrêmement utile ;
- valeurs vrai ou faux.

```dart
void main(List<String> arguments) {
  // division par 0 => short circuit
  bool choix1 = "Hello".contains("ll") || (42 / 0) == 0;
  bool choix2 = false;
}
```

Variables : Le type var

- Utilisés quand on ne souhaite pas typer une variable au moment de sa déclaration ;
- Au *runtime*, le système déterminera le type approprié ;
- Une fois déterminé, plus possible de changer.

```dart
void main(List<String> arguments) {
  var myVar = "Daniel";
  // Error: A value of type 'int' can't be assigned to a
       variable of type 'String'.
  //myVar = 2;
}
```

Variables : Le type Dynamic

- Utilisés quand on ne souhaite pas typer une variable ;
- Au *runtime*, le système déterminera à chaque affectation le type approprié ;
- Une fois déterminé, le type peut changer.

```
void main(List<String> arguments) {
  dynamic myVar = "Daniel";
  print(myVar.runtimeType);
  myVar = 2;
  print(myVar.runtimeType);
}
```

Constantes

Constantes : final

- Rend une valeur affectée immuable ;
- Elle ne pourra plus être changée ;
- C'est la référence vers la valeur qui ne peut pas changer.

```dart
void main(List<String> arguments) {
  final myVar = 1;
  //Error: Can't assign to the final variable 'myVar'.
  //myVar = 2

  final List<int> myArray;
  myArray = [1, 2];
  myArray.add(3);
}
```

Constantes : const

- Encore plus restrictif ;
- La valeur doit être connue au moment de la déclaration.

```dart
void main(List<String> arguments) {
  // Error: The const variable 'myArray' must be
      initialized.
  //const List<int> myArray;

  const List<int> myArray =  [1, 2];
}
```

Collections

Collections : Listes

- Les listes permettent de définir une collection ;
- Chaque item est associé à un index qui démarre à partir de 0 ;
- Voir null safety pour les tableaux vides ;
- Une liste sera de taille fixe ou modulaire.

```dart
void main(List<String> arguments) {
  // Error: Can't use the default List constructor.
  //List<int> listeDeNombres = new List(5);
  List<int> listeDeNombres = [1, 2, 3, 4, 5];
  listeDeNombres.add(6);
  print(listeDeNombres.length);
  listeDeNombres.remove(6);
  listeDeNombres.removeAt(0);
  listeDeNombres.sort();
  print(listeDeNombres);
}
```

Collections : Maps

- Type de collection ;
- Système de couple clé/valeur.

```
void main(List<String> arguments) {
  var pianistes = new Map();
  pianistes['Ivo Pogorelic'] = ["Chopin", "Liszt"];

  var pianiste2 = {
    'Paul Lewis': ["Chopin", "Liszt"]
  };

  pianiste2.putIfAbsent('martha argerich', () => []);
}
```

Structures de contrôle

Structures de contrôle

Toutes les structures de contrôle habituelles sont présentent :

- Les tests : `if`,`else`, `else if`, `Switch`;
- L'opérateur ternaire : `?:` ;
- les boucles : `for`,`Foreach`,`While`,`Do While`

Les fonctions

Les fonctions : Paramètres optionnels

- La fonction `main` => point d'entrée du programme ;
- les fonctions peuvent retourner des valeurs ;
- les fonctions peuvent prendre des paramètres ;
- les fonctions peuvent prendre des paramètres optionnels.

```
1 void main(List<String> arguments) {
2   print(sayHello());
3   print(sayHello("tout le monde"));
4 }
5
6 String sayHello([String name = ""]) {
7   return "Bonjour $name";
8 }
```

Les fonctions : anonymes

- Inscrites dans le code pour un besoin unique ;
- Ne pourront pas être appelées à un autre endroit.

```
1 void main(List<String> arguments) {
2   var prenoms = ["Maurizio", "Murray", "Paul"];
3   prenoms.forEach((item) {
4     print(item);
5   });
6 }
```

Les fonctions : fléchées

- Comme les fonctions anonymes ;
- Syntaxe est plus courte ;
- Une seule ligne de code ;
- `return` implicit.

```
void main(List<String> arguments) {
  var prenoms = ["Maurizio", "Murray", "Paul"];
  prenoms.forEach((item) => print(item));
}
```

Les classes

Les classes : déclaration

- Une classe par fichier (avec une majuscule) ;
- Pas de `private` à la place un `_` ;
- les `[]` rendent un paramètre optionnel ;
- On peut nommer des constructeurs `NomDeLaClasse.NomConstructeur` ;
- On peut surcharger les méthodes héritées.

Les classes : déclaration

```dart
class Personne {
  late String name;
  late int age;

  Personne(this.name, [age]) {
    age == null ? this.age = 0 : this.age = age;
  }

  Personne.empty() {
    name = "";
    age = 0;
  }

  @override
  String toString() {
    return "Nom : $name, âge : $age";
  }
}
```

Les classes : instanciation

```dart
import 'class-dec.dart';

void main(List<String> arguments) {
  Personne personne = new Personne("Hilary");
  personne.age = DateTime.now().year - 1979;

  Personne personne2 = Personne("Philippe Jaroussky",
      44);
  print(personne);
  print(personne2);
}
```

Les classes : Accesseurs et mutateurs (getters et setters)

```dart
class Personne {
  late String _name;
  late int _age;

  String get name => _name;

  set name(String value) {
    _name = value;
  }

  int get age => _age;

  set age(int value) {
    _age = value;
  }
}
```

Les classes : Héritage

Important
Pas l'héritage multiple.

```dart
import 'class-dec.dart';

class User extends Personne {
  String _email;
  User(super.name, this._email);
  String get email => _email;
  set email(String value) {
    if (value.isEmpty) {
      throw ArgumentError.value("L'email doit être
          valide");
    }
    _email = value;
  }
}
```

Les classes : Interface (classe abstraite)

Important
Le mot-clé `Interface` n'existe pas.

```dart
abstract class Human {
  late String name;
  late int age;
}
```

Les classes : Mixin - inclure une autre classe

```dart
import 'Profession.dart'; import 'class-dec.dart';

class User extends Personne with Profession {
  String _email;

  User(super.name, this._email);

  String get email => _email;

  set email(String value) {
    if (value.isEmpty) {
      throw ArgumentError.value("L'email doit être
          valide");
    }
    _email = value;
  }
}
```

Les exceptions

Les exceptions : Structure

```
1  void main(List<String> arguments) {
2    try{
3
4    }catch(e) {
5
6    }catch (e) {
7
8    }
9    finally {
10
11   }
12 }
```

Les exceptions : exceptions système

- `DefferredLoadException` : levée quand une bibliothèque différée ne se charge pas ;
- `FormatException` : levée lorsqu'une chaine de caractères ou d'autres données ne possèdent pas le format attendu et ne peuvent pas être converties ou traitées ;
- `IntegerDivisionByZeroException` : levée quand une division par zéro est tentée sur un nombre ;
- `IOException` : levée quand un problème survient lors des opérations de types IO (Input/Output) ;
- `IsolateSpawnException` : levée quand un isolate ne peut pas être créé ;
- `TimeOutException` : levée lorsqu'un délai d'expiration planifié est atteint en attendant un résultat asynchrone.

Les exceptions : Exceptions personnalisées

```
1  class ExceptionPerso implements FormatException {
2    int _source;
3
4    ExceptionPerso(this._source);
5
6    @override
7    String get message => "Le code postal doit comporter 5
         caractères !";
8
9    @override
10   int get offset => 5 - _source;
11
12   @override
13   get source => _source;
14 }
```

Traitement asynchrone

Traitement asynchrone : async et await

- L’asynchronicité est la possibilité pour une tâche d’être réalisée en parallèle d’une autre dans un laps de temps qui diffère.
- Éviter de bloquer l’application ;
- Le reste du programme doit continuer ;
- Par exemple, obtenir de l’information depuis une base de données.

Important
Deux mots-clés :`async` et `await`.

Traitement asynchrone : Future<T>

- Disponible dans le *futur*.
- Représenter une réussite ou un échec :
  - Résultat d'un calcul ;
  - Récupération de données, etc.

Traitement asynchrone : Future<T>

```dart
Future<void> main(List<String> arguments) async {
  print("Avant l'appel...");
  // On attend le retour. On pourrait ne pas le faire.
  print(await MaFonctionBonjour());
  print("Après l'appel...");
}

Future<String> MaFonctionBonjour() async {
  print("On entre dans la fonction MaFonctionBonjour");
  var traitement =
      Future.delayed(Duration(seconds: 5), () => "Le
          traitement se termine");

  print("On sort de la fonction MaFonctionBonjour");
  return traitement;
}
```