

Développement d'applications mobiles - TP

3° Bachelier en techniques graphiques

Daniel Schreurs

11 août 2022

Haute École de la Province de Liège

Table des matières i

1. Dart

- Présentation

- DartPad

2. Variables

- Les nombres

- Les chaînes de caractères

- Les booléens

- Le type var

- Le type Dynamic

3. Constantes

- final

- const

4. Collections

- Listes

- Maps

5. Structures de contrôle

6. Les fonctions

- Paramètres optionnels

- anonymes

- fléchées

7. Les classes

- déclaration

- instanciation

- Accesseurs et mutateurs (getters et setters)

Héritage

Interface (classe abstraite)

Mixin - inclure une autre classe

8. Les exceptions

Structure

exceptions système

Exceptions personnalisées

9. Traitement asynchrone

async et await

Future<T>

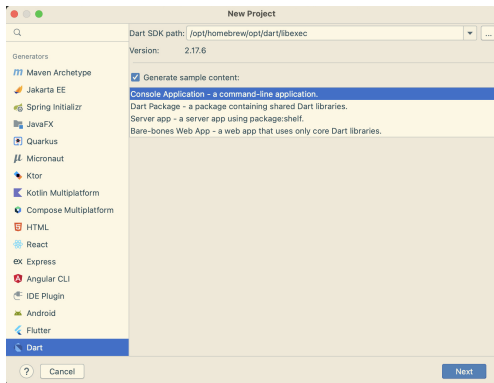
Dart

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.

```
1 void main(List<String> arguments) {  
2     // division par 0 => short circuit  
3     bool choix1 = "Hello".contains("ll") || (42 / 0) == 0;  
4     bool choix2 = false;  
5 }
```

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.

```
1 void main(List<String> arguments) {  
2     // division par 0 => short circuit  
3     bool choix1 = "Hello".contains("ll") || (42 / 0) == 0;  
4     bool choix2 = false;  
5 }
```

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.

```
1 void main(List<String> arguments) {  
2     dynamic myVar = "Daniel";  
3     print(myVar.runtimeType);  
4     myVar = 2;  
5     print(myVar.runtimeType);  
6 }
```

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.

```
1 void main(List<String> arguments) {  
2     final myVar = 1;  
3     //Error: Can't assign to the final variable 'myVar'.  
4     //myVar = 2  
5  
6     final List<int> myArray;  
7     myArray = [1, 2];  
8     myArray.add(3);  
9 }
```

Constantes : const

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

```
1 void main(List<String> arguments) {  
2     final myVar = 1;  
3     //Error: Can't assign to the final variable 'myVar'.  
4     //myVar = 2  
5  
6     final List<int> myArray;  
7     myArray = [1, 2];  
8     myArray.add(3);  
9 }
```


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.

```
1 void main(List<String> arguments) {  
2     // Error: Can't use the default List constructor.  
3     //List<int> listeDeNombres = new List(5);  
4     List<int> listeDeNombres = [1, 2, 3, 4, 5];  
5     listeDeNombres.add(6);  
6     print(listeDeNombres.length);  
7     listeDeNombres.remove(6);  
8     listeDeNombres.removeAt(0);  
9     listeDeNombres.sort();  
10    print(listeDeNombres);  
11 }
```

Collections : Maps

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

```
1 void main(List<String> arguments) {  
2     var pianistes = new Map();  
3     pianistes['Ivo Pogorelic'] = ["Chopin", "Liszt"];  
4  
5     var pianiste2 = {  
6         'Ivo ĆPogoreli': ["Chopin", "Liszt"]  
7     };  
8  
9     pianiste2.putIfAbsent('martha argerich', () => []);  
10 }
```

Structures de contrôle

Structures de contrôle

Toutes les structures de contrôle habituelles sont présentes :

- 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` implicite.

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

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

```
1 class Personne {
2     late String name;
3     late int age;
4
5     Personne(this.name, [age]) {
6         age == null ? this.age = 0 : this.age = age;
7     }
8
9     Personne.empty() {
10         name = "";
11         age = 0;
12     }
13
14     @override
15     String toString() {
16         return "Nom : $name, âge : $age";
17     }
18 }
```

Les classes : instantiation

```
1 import 'class-dec.dart';
2
3 void main(List<String> arguments) {
4     Personne personne = new Personne("Hilary");
5     personne.age = DateTime.now().year - 1979;
6
7     Personne personne2 = Personne("Philippe Jaroussky",
8         44);
9     print(personne);
10    print(personne2);
11 }
```

Les classes : Accesseurs et mutateurs (getters et setters)

```
1 class Personne {  
2     late String _name;  
3     late int _age;  
4  
5     String get name => _name;  
6  
7     set name(String value) {  
8         _name = value;  
9     }  
10  
11     int get age => _age;  
12  
13     set age(int value) {  
14         _age = value;  
15     }  
16 }
```

Les classes : Héritage

Important

Pas l'héritage multiple.

```
1 import 'class-dec.dart';
2
3 class User extends Personne {
4   String _email;
5   User(super.name, this._email);
6   String get email => _email;
7   set email(String value) {
8     if (value.isEmpty) {
9       throw ArgumentError.value("L'email doit être
10         valide");
11     }
12     _email = value;
13   }
14 }
```

Les classes : Interface (classe abstraite)

Important

Le mot-clé `Interface` n'existe pas.

```
1 abstract class Human {  
2     late String name;  
3     late int age;  
4 }
```


Les classes : Mixin - inclure une autre classe

```
1 import 'Profession.dart'; import 'class-dec.dart';
2
3 class User extends Personne with Profession {
4     String _email;
5
6     User(super.name, this._email);
7
8     String get email => _email;
9
10    set email(String value) {
11        if (value.isEmpty) {
12            throw ArgumentError.value("L'email doit être
13                valide");
14        }
15        _email = value;
16    }
17 }
```

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

- `DeferredLoadException` : levée quand une bibliothèque différée ne se charge pas ;
- `FormatException` : levée lorsqu'une chaîne 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
8         caractères !";
9
10    @override
11    int get offset => 5 - _source;
12
13    @override
14    get source => _source;
15 }
```

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>

```
1 Future<void> main(List<String> arguments) async {
2     print("Avant l'appel...");
3     // On attend le retour. On pourrait ne pas le faire.
4     print(await MaFonctionBonjour());
5     print("Après l'appel...");
6 }
7
8 Future<String> MaFonctionBonjour() async {
9     print("On entre dans la fonction MaFonctionBonjour");
10    var traitement =
11        Future.delayed(Duration(seconds: 5), () => "Le
12            traitement se termine");
13
14    print("On sort de la fonction MaFonctionBonjour");
15    return traitement;
16 }
```