Flutter: Dart

1 Introduction à Dart

Dart est un langage de programmation développé par Google. Il est utilisé principalement pour créer des applications multiplateformes avec une seule base de code. Dart est notamment le langage principal derrière le framework Flutter, utilisé pour développer des applications mobiles, web et desktop.

Les principales caractéristiques de Dart sont :

- **Orienté objet** : Dart est entièrement orienté objet.
- **Fortement typé** : Il supporte le typage statique et dynamique.
- **Multiplateforme** : Dart peut être compilé en code natif ou en JavaScript pour le web.
- **Performant** : Grâce à sa compilation en code machine, il offre des performances élevées.

2 Syntaxe de base

Un programme Dart est composé de plusieurs éléments clés :

- **Main Function** : Le point d'entrée du programme.
- **Import** : Pour inclure des bibliothèques externes.
- **Commentaires** : Utilisés pour documenter le code.

3 Syntaxe de base

Un programme Dart est composé de plusieurs éléments clés :

- **Main Function** : Le point d'entrée du programme.
- **Import** : Pour inclure des bibliothèques externes.
- **Commentaires** : Utilisés pour documenter le code.

3.1 Structure d'un programme Dart

```
// Ceci est un commentaire sur une ligne

/*
  Ceci est un
  commentaire multi-lignes
*/
import 'dart:math';  // Exemple d'importation d'une biblioth que Dart

void main() {
  print('Hello, Dart!');
}
```

3.2 Déclaration des variables

Les variables en Dart sont fortement typées et peuvent être déclarées de plusieurs manières :

- **Var** : Type inféré.
- **Dynamic** : Type dynamique.
- **Type explicite** : Type spécifié.

```
var name = "Dart";         // Type inf r   (String)
int age = 25;              // Type explicite
dynamic myVar = "Hello";   // Type dynamique
```

4 Types de données

Dart supporte plusieurs types de données :

- **Numériques** : `int` et `double`.
- **Chaînes de caractères** : `String`.
- **Booléens** : `bool`.
- **Listes** : `List`.
- **Map** : Une collection clé-valeur.
- **Set** : Une collection sans doublon.

```
int a = 10;
double b = 20.5;
String greeting = "Hello, Dart!";
bool isDartFun = true;

List<int> numbers = [1, 2, 3];
Map<String, int> ages = {"Alice": 25, "Bob": 30};
Set<String> fruits = {"Apple", "Banana", "Orange"};
```

5 Structures conditionnelles et boucles

5.1 Conditionnelles (`if`, `else`, `switch`)

```
int age = 20;

if (age >= 18) {
  print("Adult");
} else {
  print("Minor");
}

String day = "Monday";

switch (day) {
  case "Monday":
    print("Start of the week");
    break;
  case "Friday":
    print("End of the week");
    break;
  default:
    print("Midweek");
}
```

5.2 Boucles (`for`, `while`, `do-while`)

```
// Boucle for
for (int i = 0; i < 5; i++) {
  print(i);
}

// Boucle while
int i = 0;
while (i < 5) {
  print(i);
  i++;
}

// Boucle do-while
int j = 0;
do {
  print(j);
  j++;
} while (j < 5);
```

6 Fonctions et classes

6.1 Création de fonctions

Les fonctions sont définies avec le mot-clé `void` (pour les fonctions sans valeur de retour) ou avec un type de retour spécifique.

```
// Fonction sans retour
void greet() {
  print("Hello, Dart!");
}

// Fonction avec retour
int add(int a, int b) {
  return a + b;
}

void main() {
  greet();
  print(add(5, 3));  // Affiche 8
}
```

6.2 Classes et objets

Dart est un langage orienté objet. Voici comment créer une classe et un objet :

Listing 1: Exemple de classe et objet

```
class Car {
  String brand;
  int year;

  Car(this.brand, this.year);  // Constructeur

  void displayInfo() {
    print("Brand: $brand, Year: $year");
  }
}

void main() {
  Car car1 = Car("Toyota", 2020);
  car1.displayInfo();  // Affiche "Brand: Toyota, Year: 2020"
}
```

6.3 Héritage et polymorphisme

L'héritage permet à une classe d'hériter les propriétés et méthodes d'une autre classe.

```
class Animal {
  void speak() {
    print("Animal speaks");
  }
}

class Dog extends Animal {
  @override
  void speak() {
    print("Dog barks");
  }
}

void main() {
  Dog dog = Dog();
  dog.speak();  // Affiche "Dog barks"
}
```

7 Les collections

7.1 Listes (List)

Une liste est une collection ordonnée d'éléments.

```
List<int> numbers = [1, 2, 3];
numbers.add(4);   // Ajouter un    lment
numbers.removeAt(0); // Supprimer un    lment       un index
```

7.2 Maps (Map)

Une map est une collection clé-valeur.

```
Map<String, int> scores = {"Alice": 100, "Bob": 80};
scores["Charlie"] = 90;  // Ajouter une cl
print(scores["Alice"]);   // Affiche 100
```

7.3 Sets (Set)

Un set est une collection non ordonnée sans doublons.

```
Set<String> fruits = {"Apple", "Banana", "Orange"};
fruits.add("Grapes");     // Ajouter un    lment
fruits.remove("Banana");  // Supprimer un    lment
```

8 Futures, async/await

8.1 Futures

Un `Future` représente un résultat qui n'est pas encore disponible, mais le sera dans le futur.

```
Future<String> fetchData() async {
  await Future.delayed(Duration(seconds: 2));  // Simule un d lai
  return "Data fetched!";
}

void main() async {
  String data = await fetchData();  // Attendre le r sultat
  print(data);  // Affiche "Data fetched!"
}
```

8.2 Async et Await

Les mots-clés `async` et `await` permettent d'écrire du code asynchrone de manière lisible.

```
Future<void> fetchData() async {
  print("Fetching data...");
  await Future.delayed(Duration(seconds: 2));
  print("Data fetched!");
}

void main() async {
  await fetchData();
}
```