Présentation du Langage Dart

Par Maxence CUGUEN

Etude en profondeur de Dart

Expliquer sa création

Appliquer les différentes notions abordées en cours sur ce langage

Identifier les différences avec les autres langages de POO et notamment Java

Problématique

Introduction

- Langage développé par Google
- Utilisé pour :
  - Développement Web Côté Client
  - Développement d'applications mobile
  - Développement côté serveur
- - Crée pour remplacer JavaScript

Historique

V1 dévoilée au Grand Public en Octobre 2011.

Réalisé par Lars Bak et Kasper Lund, développeurs chez google.

Anciennement nommé Dash.

Devenu une norme ECMA(la 408) en juin 2014

Idée d’intégrer Dart dans Chrome abandonnée en 2015

Dart et JavaScript

Prévu pour concurrencer JavaScript et résoudre certains problèmes de celui-ci

Meilleurs Performances que JS, adapté aux gros projets,

Système de classes (absent dans JS)

Dart devait être implémenté dans Chrome -> Idée annulée

Google sors Dartium, variance de Chrome avec VM Dart incluse

Maintenant Dart est compilable en JS

La Compilation

Dart2js : Le transcompilateur officiel de Dart pour le déploiement, compilateur source à source, qui prend du code Dart et le compile en JS déployable.

Dartdevc : Le Transcompilateur officiel de Dart pour le développement seulement. Il compile lui aussi du code Dart en JS

Typage Optionnel

Typage Optionnel

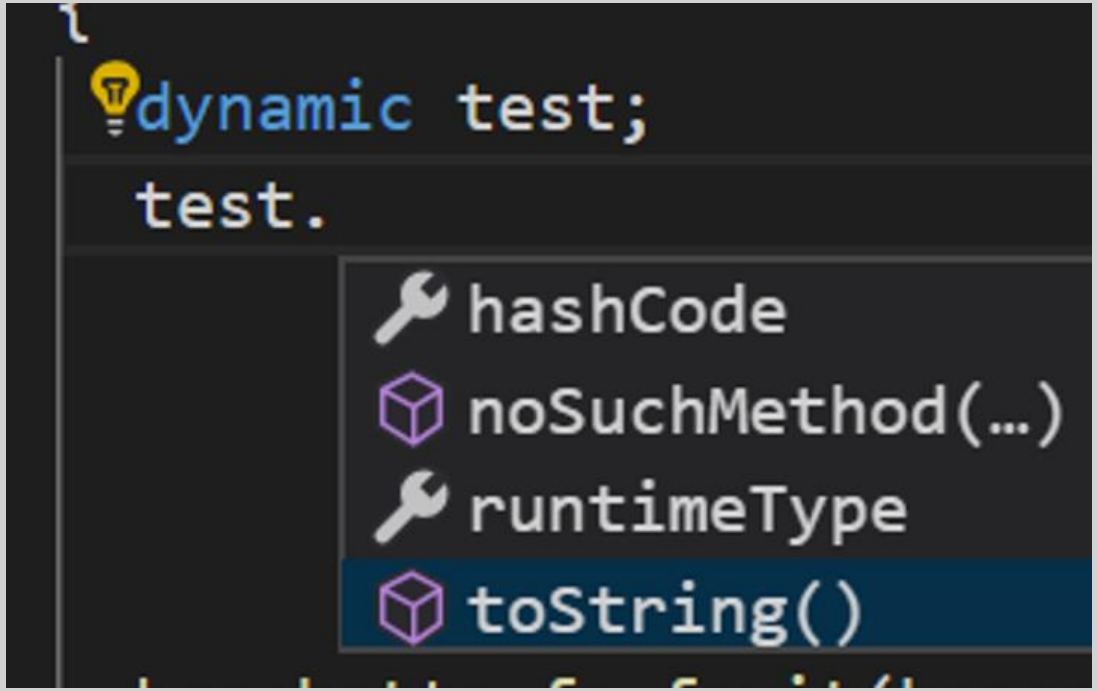
- Statique et Dynamique ?
- Mot clé var
- Type dynamic → pas de warnings / erreurs statiques. Le système suppose que toute propriété existe
- Pas un langage Pur à Objet (types primitifs présents)

```
var foo;
foo.foo();
dynamic bar;
bar.bar();
```

```
toto (tata) {
  return tata;
}
```

Le type Dynamic

- Hérite de la classe Object
- Tant qu'une var n'a pas de type elle est dynamic
- Différence avec Object : dynamic peut faire croire que toute propriété existe, pas object



Application

```
void compte_sloubifuit(brochette)
{
  int i=0;
  while(brochette.debrocher() != null) i++;
  print("Il y avait " + i.toString() + " fruits");
}
```

- La méthode compte_sloubifuit() attend un paramètre sans savoir son type.
- Brochette.debrocher() fonctionne statiquement car brochette est de type dynamic.

Héritage Multiple et Mixins

Héritage Multiple et Mixins

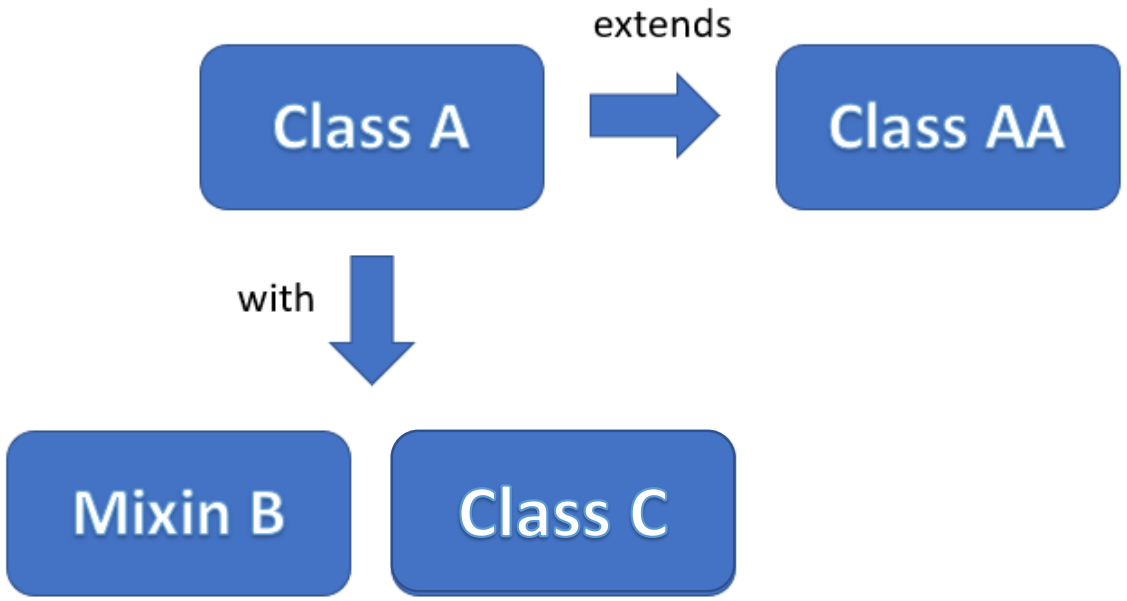
Héritage Multiple non disponible en Dart

Héritage Multiple =/= Mixins

Concept des Mixins

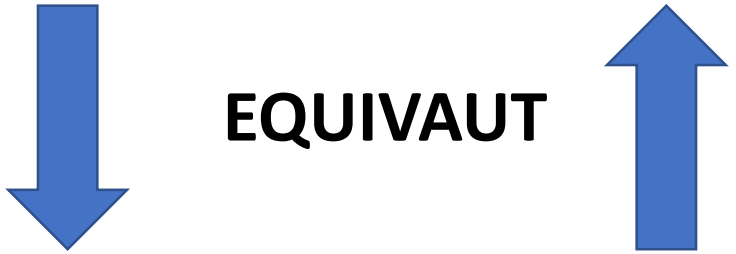
- Elles peuvent être utilisées pour offrir des propriétés à des classes sans passer par de l'héritage simple
- Elles ressemblent aux des classes abstraites :
  - Elles sont non instanciables
  - Elles peuvent contenir des définitions de méthodes ou seulement leur déclaration
- Mais l'on ne peux pas hériter d'une mixin

```dart
class A extends AA with B, C {}
mixin B {}
mixin C {}
class AA {}
```



Mixins et Héritage

```
class A extends AA with B, C {}
mixin B {}
class C {}
class AA {}
```




```
class A extends AA with B, C {}
mixin B {}
mixin C {}
class AA {}
```

Mixins et Spécificité

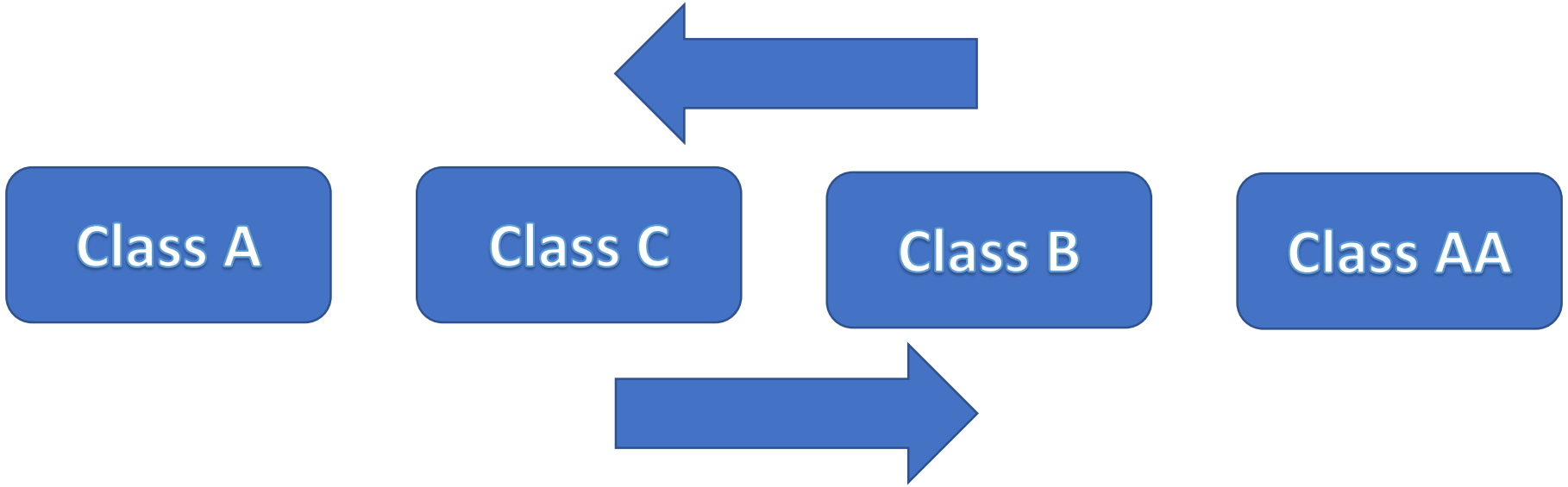
```
class A extends AA with B, C {}
mixin B {
  foo() {
    print("B");
  }
}
class C {
  foo() {
    print("C");
  }
}
class AA {
  foo() {
    print("AA");
  }
}
```

```
A a = new A();
a.foo();
```

OUTPUT : C

```dart
class A extends AA with B, C {}
mixin B {}
class C {}
class AA {}
```

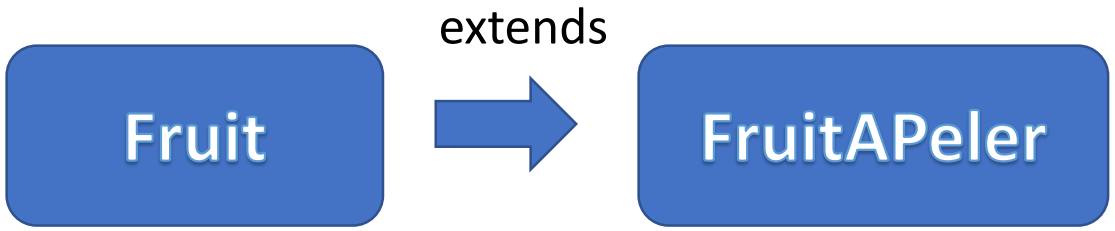
PLUS SPECIFIQUE



MOINS SPECIFIQUE

Casts et Assignement Compatibility

Casts et Assignement Compatibility




```
List<Fruit> fruits = new List<Fruit>();
Fruit debrocher()
{
  Fruit fruit =  fruits.last;
  fruits.removeLast();
  return fruit;
}
```

Classe FruitAPeler hérite de Fruit

Méthode debrocher() a un type de retour Fruit.

Casts et Assignement Compatibility

Code Java

```java
private static void exemple_banane()
{
    Brochette brochette = new Brochette();
    Banane banane = new Banane();
    brochette.embrocher(banane);
    FruitAPeler f = (FruitAPeler) brochette.debrocher();
    f.peler();
}
```

Code Dart

```dart
void exemple_banane() {
  Brochette brochette = new Brochette();
  Banane banane = new Banane();
  brochette.embrocher(banane);
  FruitAPeler f = brochette.debrocher();
  f.peler();
}
```

Assignement Compatibility en SuperType.

- En java : Assignement est jugé safe d'un sous-type à un super-type.

- En dart :
  - Assignement est jugé safe d'un sous-type à un super-type.
  - Assignement est jugé safe d'un super-type à un sous-type.

-> Le Cast devient donc inutile.

-> Pas d'erreur statique, mais si au runtime le fruit retourné n'est pas un super-type de FruitAPeler (ou un FruitAPeler), on aura une erreur dynamique (exemple si le fruit est une Fraise)

Metaprogrammation

Métaprogrammation : une question de réflexion

```
import 'dart:mirrors';
```

Pas de classe Classe, pour créer une nouvelle instance on utilise la méthode newInstance() de ClassMirror

Cette librairie propose par exemple :

- MirrorSystem
- ClassMirror
- InstanceMirror
- LibraryMirror

- Reflexion : Mécanisme qui permets d’avoir et d’utiliser au runtime des metadata sur des classes.

- Symbol : C’est un Objet, qui s’apparente à une string qui permet de faire de la reflection sur les librairies. Ils font office d’intervalle entre les strings (lisible par l’humain) et les machines.

Conclusion

- Dart est un langage simple à appréhender, notamment pour un développeur Java du à sa syntaxe similaire.
- Le typage optionnel et les mixins offre beaucoup de liberté au développeur.
- Malgré que l'objectif de le faire renverser JavaScript ai été abandonné, il reste meilleur que lui sur beaucoup de points.
- Dart a un avenir prometteur (utilisé sur bon nombre de projets chez Google).