LEE : Tutoriel de tour de magie

Comment faire disparaitre quelque chose avec la formule magique

Scala Implicit

Nafise ABDOULAZISE

Introduction

Dans ce tutoriel nous verrons comment utiliser la notion de scala implicit et comment maitriser ce tour de magie. Pour cela nous allons nous intéresser sur l'exemple suivant :

```
class TourMagie(s:String) {
  def disparition(animal: String): Unit = {
    if (s == " a disparu")
      println(animal+s)
    else None
  }
  def metamorphose(animal: String): Unit = {
    if (s == " a ete metamorphose")
      println(animal+s)
    else None
  }

}

val magie1 = new TourMagie(" a disparu")
val magie2 = new TourMagie(" a ete metamorphose")


magie1.disparition( animal = "lapin")
magie2.metamorphose( animal = "colombe")
```

Dans cet exemple il y a une classe TourMagie qui contient 2 méthodes :

- Disparition qui affiche un message
- Metamorphose qui affiche un message

Voici le résultat :

```
lapin a disparu
colombe a ete metamorphose
```

Nous allons utiliser ce code pour la suite du tutoriel.

Les paramètres implicit

Nous allons maintenant s'attaqué à notre premier tour de magie.

```
magie1.disparition
magie2.metamorphose( animal = "colombe")
```

OÙ EST PASSÉ LE LAPIN ???

Si on exécute le code suivant vous allez peut-être penser que la méthode disparition ne va pas marcher car elle n'a pas de paramètre. Mais n'oubliez pas que vous êtes dans un tutoriel de magie. L'exécution nous donne l'affichage suivant :

Abracadabra Scala Implicit !!!!!

```
lapin a disparu
colombe a ete metamorphose
```

On voit que l'exécution s'est déroulée sans aucun soucis.
Hein ?? Quoi ?? Comment c'est possible ?? J'y arrive. Comme pour tous les tours de magie le nôtre nécessite bien sûr quelques préparations en amont.
Il est vrai qu'on ne donne pas le paramètre directement mais on le donne indirectement.

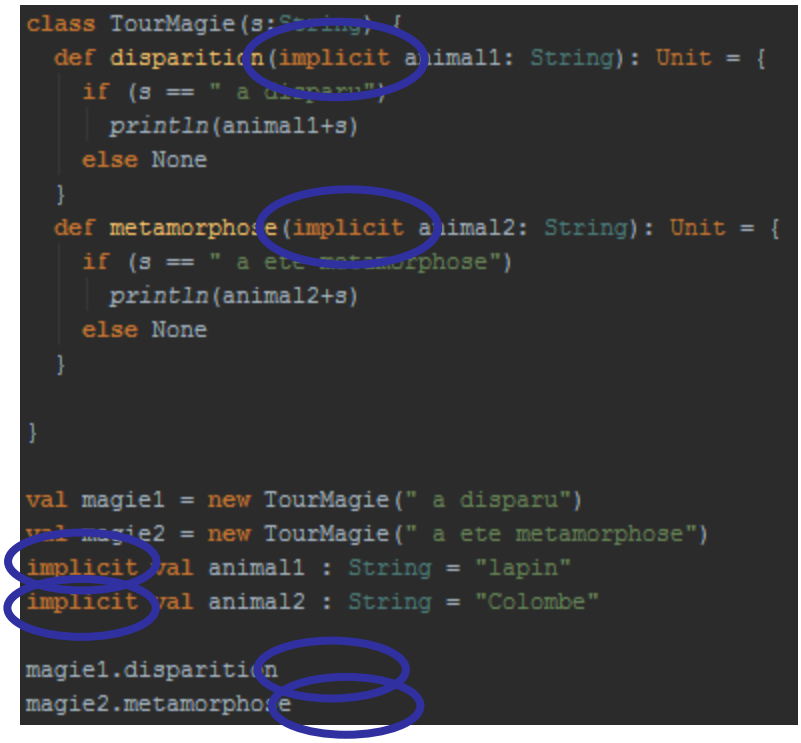
```
class TourMagie(s:String) {
  def disparition(implicit animal: String): Unit = {
    if (s == " a disparu")
      println(animal+s)
    else None
  }
  def metamorphose(animal: String): Unit = {
    if (s == " a ete metamorphose")
      println(animal+s)
    else None
  }

}

val magie1 = new TourMagie(" a disparu")
val magie2 = new TourMagie(" a ete metamorphose")
implicit val animal : String = "lapin"

magie1.disparition
magie2.metamorphose( animal = "colombe")
```

Il est donc nécessaire d'implémenter la méthode en précisant que le paramètre attendu est implicit et il faut définir ce paramètre préalablement.
Le compilateur voit que la méthode disparition a besoin d'un paramètre implicit de type String. Il va prendre comme paramètre la variable implicit qui a été précédemment déclaré et qui correspond au type attendu.
Mais qu'est ce qu'il se passe s'il y a plusieurs variables implicit du même type ???



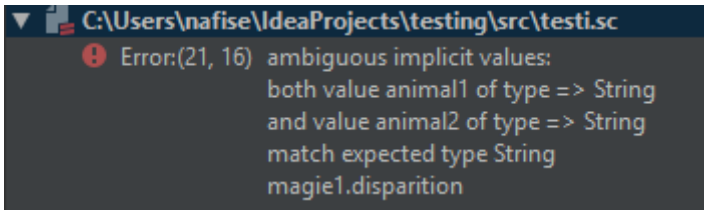
```
class TourMagie(s:String) {
  def disparition(implicit animal1: String): Unit = {
    if (s == " a disparu")
      println(animal1+s)
    else None
  }
  def metamorphose(implicit animal2: String): Unit = {
    if (s == " a ete metamorphose")
      println(animal2+s)
    else None
  }

}

val magie1 = new TourMagie(" a disparu")
val magie2 = new TourMagie(" a ete metamorphose")
implicit val animal1 : String = "lapin"
implicit val animal2 : String = "Colombe"

magie1.disparition
magie2.metamorphose
```

Dans cette situation le compilateur remarque une ambiguïté et ne compile pas.



```
C:\Users\nafise\IdeaProjects\testing\src\testi.sc
  Error:(21, 16) ambiguous implicit values:
                 both value animal1 of type => String
                 and value animal2 of type => String
                 match expected type String
                 magie1.disparition
```

Les classes implicit

En général dans un langage orienté objet, pour créer une instance d'une classe il faut créer une variable et faire un « new » de cette classe avec les paramètres nécessaires. C'est exactement ce qui a été fait précédemment. Mais on exécute le code suivant :

```
" a disparu".disparition
```

Ah oui j'ai failli oublier

Abracadabra Scala Implicit !!!!!

On obtient le résultat suivant

```
lapin a disparu
```

Whaaaaaouuuh mais comment ça marche ????

L'heure des révélations a sonné :

```
implicit class TourMagie(s:String) {
  def disparition(implicit animal1: String): Unit = {
    if (s == " a disparu")
      println(animal1+s)
    else None
  }
  def metamorphose(implicit animal2: String): Unit = {
    if (s == " a ete metamorphose")
      println(animal2+s)
    else None
  }

}

implicit val animal1 : String = "lapin"

" a disparu".disparition
```

Il nous suffit pour cela de déclarer notre classe comme étant une classe implicit et le tout est joué. On peut donc se permettre d'utiliser une méthode d'une classe sans créer directement une instance de cette classe.

Les méthodes implicit

On arrive maintenant au dernier tour de magie. Nous venons de voir que pour ne pas à avoir créer une instance d'une classe on doit définir la classe en tant que classe implicit. Je vous dis maintenant qu'il est possible d'avoir le même résultat sans déclarer une classe implicit.

```
class TourMagie(s:String) {
  def disparition( implicit animal1: String): Unit = {
    if (s == " a disparu")
      println(animal1 + s)
    else None
  }

  def metamorphose(animal2: String): Unit = {
    if (s == " a ete metamorphose")
      println(animal2 + s)
    else None
  }
}

implicit val test : String = "lapin"

" a disparu".disparition
```

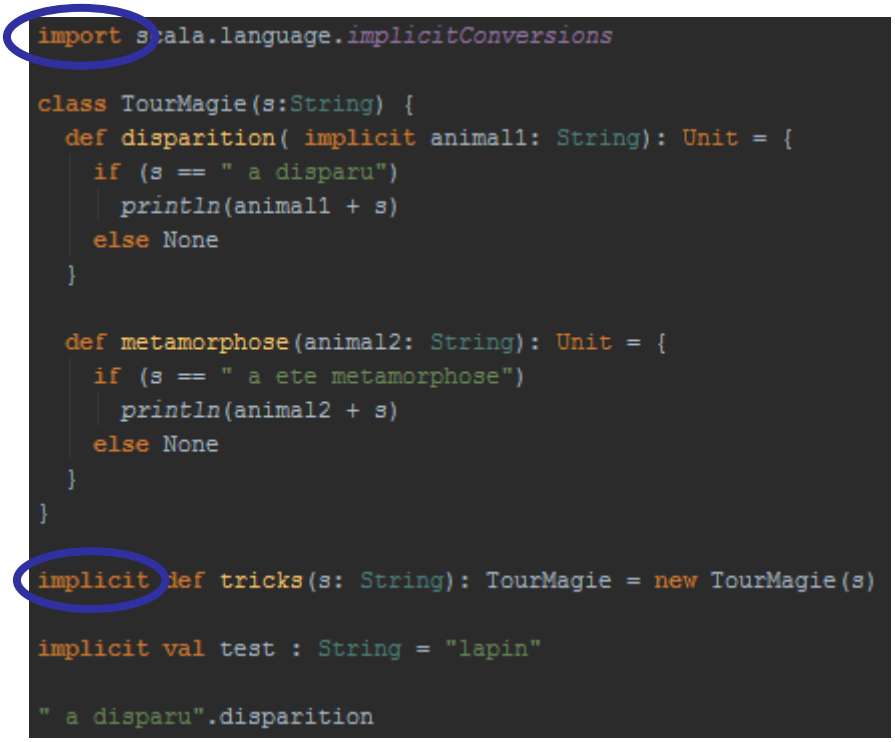
Abracadabra Scala Implicit !!!!!

Avec ce code, j'obtiens le résultat suivant

```
lapin a disparu
```

WHAAAAT ???????

Bon je l'avoue je ne vous ai pas tout montré. C'est bien ça le but de la magie n'est-ce pas ?
Dans le code précédent il manque juste ces deux lignes :



```scala
import scala.language.implicitConversions

class TourMagie(s:String) {
  def disparition( implicit animal1: String): Unit = {
    if (s == " a disparu")
      println(animal1 + s)
    else None
  }

  def metamorphose(animal2: String): Unit = {
    if (s == " a ete metamorphose")
      println(animal2 + s)
    else None
  }
}

implicit def tricks(s: String): TourMagie = new TourMagie(s)

implicit val test : String = "lapin"

" a disparu".disparition
```

Ici, la fonction tricks est un pont entre le type String et le type TourMagie. Sachant qu'il est défini par implicit, on a plus besoin de déclarer la classe par implicit ce qui ne nous empêche pas d'utiliser les méthodes qui y ont été défini.

L'utilité

En gros scala implicit permet de laisser au compilateur d'aller chercher lui-même les éléments qui lui manque au lieu que ça soit le développer qui l'indique.
L'utilisation de scala implicit peut être très utile lorsque vous avez besoin d'affecter la même valeur à plusieurs reprises. Cela vous permet aussi d'avoir une meilleure lisibilité du code. Les bibliothèques de scala utilisent souvent scala implicit pour définir des implémentations par défaut.
Il y a aussi des inconvénients avec cet outils. Malgré le fait qu'il facilite la lecture d'un code, déboguer un code devient assez compliqué.