Chapitre 2 Scala Orienté Objet

Sommaire

1. Case class vs class simple
2. Héritage Simple
3. Trait vs Abstract
4. Object Singleton
5. org.apache.camel.archetypes:camel-archetype-scala 2.12.11

Programmation orientée objet

Case class vs class simple

Une **Classe**: est un modèle de définition pour des objets ayant le même ensemble d'attributs, et le même ensemble d'opérations

En scala nous avons 2 types de class , les **class simples** et les **case class**

Case class vs class simple

class PersonSimple(nom:String)

case class PersonCase(nom:String)

1 Instance

```
val p1=new PersonSimple( "fall")
val p2=new PersonsSimple( "fall")
 val p3=PersonCase( "diop")
 val p4=PersonCase( "diop")
```

2 Accès

```
p1.nom                    p3.nom
```

3 Comparaison

```
p1.equals(p2)             p3.equals(p4)
```

4 Copy

```
val p5=p1.copy()          val p5=p3.copy()
```

5 Patern matching

```
def  guesseth(p:PersonCase ):String={
        p match {
        case PersonCase("diouf")|PersonCase("sene" ) => "serere"
        case PersonCase("ba")|PersonCase("sow") =>"peul"
        case _ => "dont know"
        }
}
```

Conclusion case class vs class simple

Case class VS Class simple

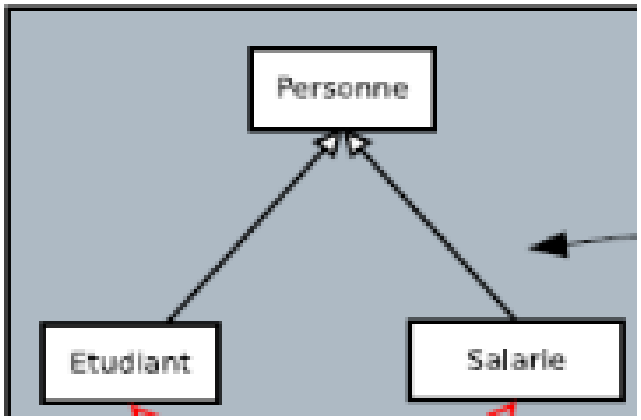
	Case class	Class simple
new pour instancier	no	yes
pattern matching	yes	no
comparer des instance	yes	no
faire des copies	yes	no
accéder direct aux attribut sans personnaliser la class avec var ou val	yes	no
lourd	++	++

Héritage

En scala nous avons trois types héritage

- Héritage Mono simple
- Héritage Mono avec trait ou class abstraite
- Héritage multiple

11 Héritage Mono simple




```
package POO_MSD_Heritage_Simple

class Person(nom:String, age:Int){

  def presentation()=s" my name is $nom and i am $age"

  def manger()="je mange"


  final def checkmaturity():Unit={
    if(age<=25)
      println("je ne suis pas mature")
    else
      println("je suis mature")
  }
}
```

```scala
package POO_MSD_Heritage_Simple

class Etudiant(nom:String, age:Int, universite:String) extends Person(nom, age){

  override def presentation()=s" my name is $nom , i am $age and we come  from $universite"

  override def manger()=s"${super.manger} dans les resto U"
}
```

```scala
package POO_MSD_Heritage_Simple

class Salarie(nom:String, age:Int, salaire:Int, epargne:Int) extends Person(nom, age){

  def argentdispo()=s"La somme dispo est ${salaire+epargne}"

  override def manger()=s"${super.manger} un repas que ne depasse pas ${salaire+epargne}"



}
```

11 Héritage Mono simple

Une case class ne peut pas hériter une case class

Une case peut hériter une class simple

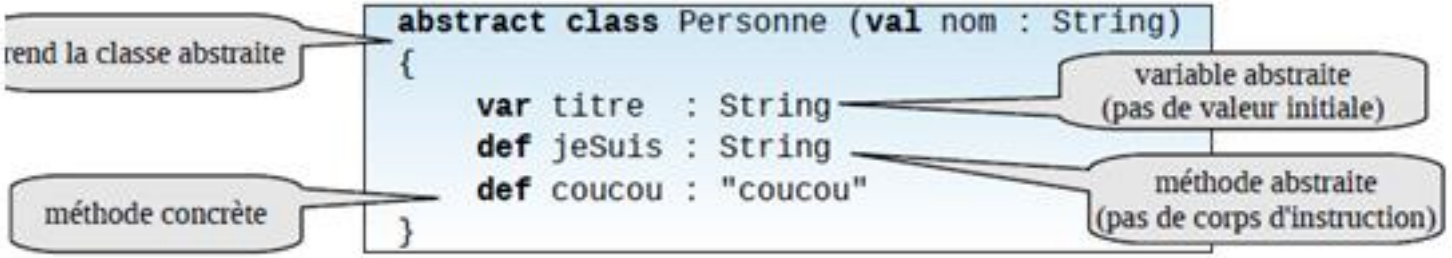
Une class simple peut hériter une case class

Une class simple peut hériter une class simple

12 Héritage Mono [class abstract & trait)

- Une classe abstraite est une class dont l'implémentation n'est pas complète et qui n'est pas instanciable, elle sert de base à :
  - **Factoriser le code**
  - **Gérer les contraintes**
- Les traits sont comme les interfaces en Java

12 Abstract



Impossible de créer des instances d'une classe abstraite

```
val p = new Personne              // !! erreur de compilation
```

12 Abstract:Application1

```
package POO_Abstract

abstract class  Compte{
  var id:Int
  var solde:Int

  def retire(somme:Int):Unit={
    solde=solde-somme
  }

  def depot(somme:Int):Unit={
    solde=solde+somme
  }

  def afficheInfoCompt():String

}
```

```scala
package POO_Abstract

case class CompteBancaire(var id:Int, var  solde:Int,
                          nomclient:String,idagence:Int)
  extends Compte{

  def afficheInfoCompt()=s"nom:$nomclient id:$id agence:$idagence solde:$solde"
}
```

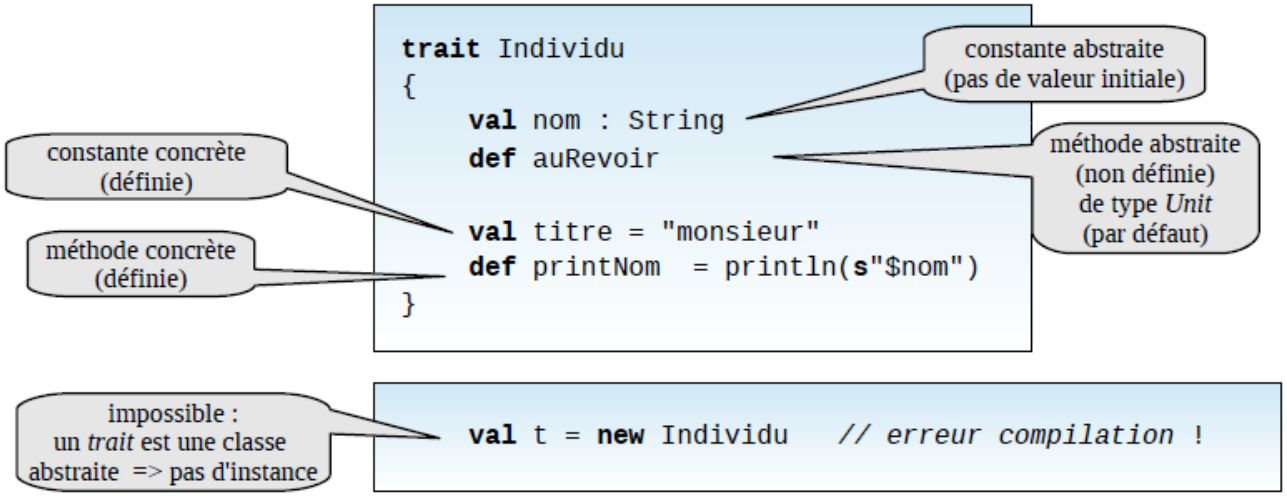
```scala
package POO_Abstract

case class OrangeMoney(var id:Int, var  solde:Int
                       ,nomclient:String )
  extends Compte {

  def afficheInfoCompt()=s"nom:$nomclient telephone:$id solde:$solde"

  def changernumero(newnumber:Int) :Unit =id=newnumber
}
```

Abstract: Factoriser un code

```
def showId(compte: Compte): String = {
  compte match {
    case CompteBancaire(id, _, _, _) => s"Id de votre compte est $id"
    case OrangeMoney(id, _, _) => s"Votre numero de  tel est $id"
    case _ => "Invalid action"
  }
}
```

Un trait est une classe abstraite sans paramètre

Les membres abstraits ne peuvent pas être privés



Un trait est semblable à une interface java (en nettement moins restrictif)

Trait Vs class abstract

- Constructeur

trait Person(nom:String) **abstract class Person(nom:String)**

- Ajout dans une instance

```
trait Traitement1 { def action = " i do something "}

abstract class Traitement2  { def action = " i do something "}

case class Ordinateur(marque:String)

val ordi1=new Ordinateur(" HP " ) with Traitement1  val ordi2=new Ordinateur(" HP " ) with Traitement2
ordi.action
```

Héritage Multiple

- Pour faire de l'héritage multiple , il faut utiliser le mot with

NB:with est toujours suivi d'un type trait

Héritier(e)	extends	with
case class(..) class simple trait class abstract	case class(..) si la class à hérité n est pas une class class simple trait class abstract	trait

trait T1; trait T2 ; trait T3 | abstract class Abs | class C1 | case class C2(..)

class C extends Abs1 with C2 => error

class C extends Abs1 with T1 with T2 with T2 => ok

Trait Vs Abs

Action	Trait	Class Abstract
Ajouter dans une instance	?	?
Avoir un constructeur	?	?
with class abstract with trait	?	?
Faire appel à des fonction java sans wrapper(lib)	?	?

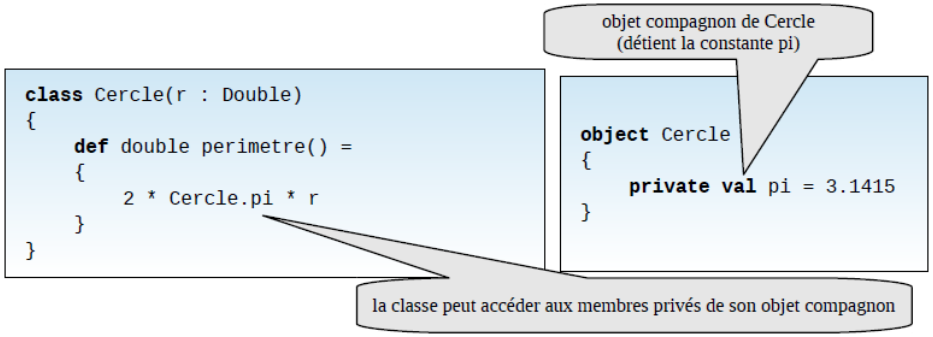
13 Singleton

Le langage Scala ne permet pas de définir des **méthodes ou des attributs statiques** dans une classe, pour faire cela, il faut utiliser un type de classe particulier qui est instancié par défaut et ne peut être réinstancié. Scala utilise le nom de «singleton object» pour définir ce type de classe. La définition d'un singleton object est similaire à la définition d'une classe, sauf qu'il faut utiliser le mot-clé «object» à la place du mot-clé «class»

```
object Logger {
  def info(message: String): Unit = println(s"INFO: $message")
}
```

Objet compagnon

Un *object* du même nom qu'une classe, conçu pour détenir des attributs et méthodes (statiques) qui n'ont pas besoin des attributs de la classe



L'objet compagnon et la classe doivent être dans le même fichier