TD Si2 FIP - Groupe IE4 (A/B)

Exo 1:

Exercice 1 (MinMax*). Definir (spécification + code) en Scala une fonction minMax qui,´etant donnes deux entiers i et j, retourne un doublet contenant le minimum puis le maximum (dans cet ordre) de i et j.

- ajouter "quelconque" dans la spec des params pour insister
- spécification ressemble trop au code : comment exprimer ca differemment ? "Ordre croissant", ou même vous baser sur l'enonce.

```
/**
  * @param i un entier
  * @param j un entier
  * @return le tuple de i,j ou i<j , ou (j,i) et j<=i
  */
 def minMax(i: Int,j: Int): (Int,Int) = {
   if (i<j) (i,j)
   else (j,i)
 }
```

- Remarque cas particulier: valide quand i = j: minMax(3,3) ~> (3,3)
- Bonne question!

(i < j) match { // type booléen -> motifs pour les booléens
case true => (i,j)
case false => (j,i)
}

Exo 2:

On souhaite modéliser des données représentant des personnes. Pour chaque personne, on ne considère que

- son nom - Demange
- son prénom - Delphine
- son année de naissance - 1998

Definir les types alias type Nom et type Annee modélisant les noms et prénoms, ainsi que les années.

- type alias : juste un autre nom pour un type qui existe déjà

Type alias qui utilise des types tuples -> pas de nommage des composantes et uniquemenent ._1, ._2

```
type Nom = (String, String) // Nom de famille, prenom
type Annee = Int // pas toute la date de naissance, que l'annee
```

-

Definir le type alias type Personne en utilisant des tuples : meme technique que precedemment

```
type Personne = (Nom, Annee) // type doublet (tuple taille 2)

val p1: Personne = (("Delphine", "Demange"), 1998)
val p2: Personne = (("Vincent", "Lechevalier"), 2001)
val p3: Personne = ("Delphine", "Demange", 1998) // Marche pas!
```

Variante exercice 4 : Personne par un type algébrique.

- avec alternative : case class, sealed trait, extends
- sans alternative : case class

Solution proposee:

```

sealed trait Personne // type algebrique mais NE CONVIENT PAS
 case class Nom extends Personne
 case class Annee extends Personne
```

Etre une personne :

- soit être un nom
- soit être une année

```
type Nom = (String, String)
sealed trait Personne
case class P(prenomNom: Nom, annee: Annee) extends Personne

val p1: Personne = P( ("Delphine", "Demange") , 1998)
```

Pas donner le même nom au type et au type+constructeur de la case class : trouver des idées de nom :-)

```
case class Personne(prenomNom: Nom, annee: Annee)

val p1: Personne = Personne(("Delphine", "Demange") , 1998)
// Attention : ne pas oublier le nom du constructeur
// 1ere Personne : nom du type de la val
// 2eme Personne : le constructeur du type Personne

```

Personne modelisee avec des tuples :

Attention : specification du return : mentionner les parametres pour expliquer le resultat

Annee de majorite : annee de naissance + 18 ans

```
  /**
    * @param p une personne
    * @return  l'annee où la personne p devient majeure (18 ans)
    */
  def anneeMajorite(p: Personne) : Annee = {

       p._2+18


  }

    /**
   *@param p une personne
   *@param a une annee, posterieure à l'annee de naissance de p
   *@return  l'age de la personne p, en l'annee a
   *         - L'age d'une personne nee en 2001 en 2006 : 5 ans
   *         - L'age d'une personne nee en 2001 en 1993 : ???
   */

  def age(p: Personne, a: Annee) : Int = {

      a-p._2 // toujours un entier positif ou nul


    }
```

```scala

L'exercice impose la signature de la fonction age, mais
en general, attention, il peut y avoir (plein) d'autres solutions.

Exemple : Avec le type option pour le resultat

   /**
   *@param p une personne
   *@param a une annee quelconque
   *@return  l'age de la personne p, en l'annee a, si cela a du sens
   *         - L'age d'une personne nee en 2001 en 2006 : 5 ans
   *         - L'age d'une personne nee en 2001 en 1993 : pas de sens
   */

  def age(p: Personne, a: Annee) : Option[Int] = {

    if (a >= p._2) {
      Some(a-p._2)
    } else {
      None
    }
   }
```

Exercice 3

Animaux : 3 cas possibles, tous distincts, AVEC ALTERNATIVES

- vache
- chien
- lion

```
sealed trait Animal // Type de tous les animaux possibles
case object Vache extends Animal
case class Chien extends Animal
case object Lion extends Animal

val a1 : Animal = Vache
val a2: Animal = Lion
val a3: Animal = Chien
```