**Durée du CC : 1h15**

Prénom ..... Nom : .....
Toutes les méthodes demandées sont testées dans le `main()`.

Dans ce sujet, on s'intéresse à une population de sangliers et de geais dans une forêt de Loire-Atlantique. Pour les représenter, on définit une classe abstraite `Animal`, et deux case classes `Sanglier` et `Geai` qui en héritent. Dans cette modélisation simple, les animaux possèdent un seul attribut : leur `poids`, en kg.

**Question 1** À l'évidence, ces deux espèces ne sont pas les seuls animaux habitant la forêt. À quoi peut-on reconnaître cela dans le code ?

Dans la suite, on fera attention à ce que tous les pattern matching fonctionnent même si d'autres espèces animales sont intégrées au code.

**Question 2** Chaque animal a besoin d'un certain nombre de chênes pour vivre. La règle est la suivante [1] :

- pour survivre, un `Sanglier` a besoin de manger une quantité de glands dépendant de son poids : il lui faut `10` chênes par kilogramme,
- un `Geai` a besoin de `10` chênes pour nicher, peu importe son poids,
- les autres animaux n'ont pas spécifiquement besoin de chênes pour vivre (c'est-à-dire qu'ils ont besoin de `0` chêne).

Implémenter la méthode `nbChenes()` de la classe `Animal` (ligne 5) : elle doit renvoyer le nombre de chênes dont l'`Animal` a besoin pour vivre.

**Question 3** Implémenter, **via un pattern matching**, la méthode `listeGeais()` (ligne 35), qui renvoie la sous-liste des geais parmi une liste d'animaux. L'ordre des geais ne doit pas être modifié.

**Question 4** Implémenter, **en une ligne et sans pattern matching**, la méthode `listeQuintal()` (ligne 39), qui renvoie la sous-liste (sans en modifier l'ordre) des animaux pesant au moins `100`kg.

**Question 5** Implémenter la méthode `plusLourd()` (ligne 43) qui renvoie l'`Animal` le plus lourd dans une liste. Attention, cette méthode doit renvoyer une `Option[Animal]` : `None` si la liste est vide, ou `Some(a)` si elle est non-vide et que `a` en est l'animal le plus lourd.

**Question 6** Implémenter, **en utilisant foldLeft()**, la méthode `nbChenesTotal()` (ligne 47) qui renvoie le nombre total de chênes dont a besoin une population (c'est-à-dire, une liste) d'animaux.

1. Ces chiffres sont évidemment fictifs.

L'État décide de raser une partie de la forêt dans l'objectif de construire un aéroport sur son emplacement. On cherche à déterminer quel va être l'impact de cette artificialisation sur la population animale [2].

On va modéliser la situation de manière exagérément simple : étant donnée une population animale (sous forme de `List[Animal]`) et un nombre de chênes restants (sous forme d'`Int`), on va énumérer les animaux dans l'ordre de la liste. Chaque animal `a` a besoin d'un nombre de chênes égal à `a.nbChenes()` pour survivre. S'il reste suffisamment de chênes, alors `a` survit, mais enlève `a.nbChenes()` chênes à la réserve disponible. Autrement, `a` meurt (mais n'enlève pas de chênes à la réserve).

**Question 7** Implémenter, **via un pattern matching**, la méthode `survivants()` (ligne 51) qui détermine la liste des animaux survivants. Cette méthode prend deux paramètres : une population d'animaux et le nombre de chênes restants après l'implantation de l'aéroport. L'ordre des animaux dans la liste doit être conservé.

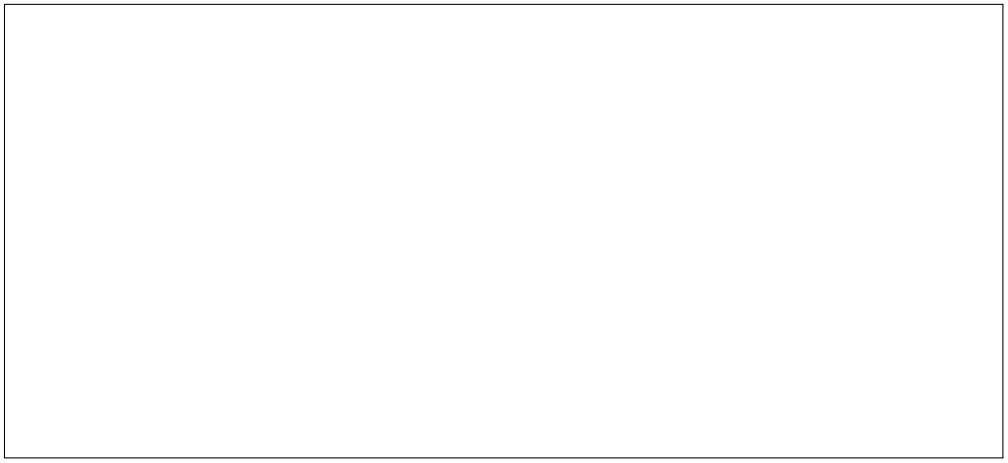
**Question 8** (☠) Implémenter, **en utilisant foldLeft()** et sans appeler `survivants()`, la méthode `morts()` (ligne 55) qui détermine la liste des animaux qui ne survivront pas à la déforestation. Cette méthode prend deux paramètres : une population d'animaux et le nombre de chênes restants après l'implantation de l'aéroport. Cette méthode n'a pas besoin de conserver l'ordre des animaux dans la liste initiale.

2. L'artificialisation des sols représente la première cause de l'effondrement de la biodiversité. Depuis 1970, **69%** de la population de vertébrés a disparu de la surface du globe. Ces chiffres, eux, sont malheureusement bien réels (`https://www.wwf.fr/sites/default/files/doc-2022-10/LPR%202022%20VFINAL_Page_pageBD.pdf`).

```scala

  abstract class Animal {
    val poids : Int

    def nbChenes() : Int = {
```



```scala
    }
  }

  case class Sanglier(poids : Int) extends Animal
  case class Geai(poids : Int) extends Animal

  object Main {

    def main(args : Array[String]) : Unit = {

      println(Sanglier(105).nbChenes())
          // 1050
      val l=List(Sanglier(90),Sanglier(150),Geai(1),Geai(2),Sanglier(120))
      println(listeGeais(l))
          // List(Geai(1),Geai(2))
      println(listeQuintal(l))
          // List(Sanglier(150), Sanglier(120))
      println(plusLourd(l))
          // Some(Sanglier(150))
      println(nbChenesTotal(l))
          // 3620 (=90*10+150*10+10+10+120*10)
      println(survivants(l,2000))
          // List(Sanglier(90), Geai(1), Geai(2))
      println(morts(l,2000))
          // List(Sanglier(120), Sanglier(150))
          // ou List(Sanglier(150), Sanglier(120))
    }
```

```
34
35      def listeGeais(l : List[Animal]) : List[Geai] = {
36
37      }
38
39      def listeQuintal(l : List[Animal]) : List[Animal] = {
40
41      }
42
43      def plusLourd(l : List[Animal]) : Option[Animal] = {
44
45      }
46
```

```
47      def nbChenesTotal(l : List[Animal]) : Int = {
48
49      }
50
51      def survivants(l : List[Animal], n : Int) : List[Animal] = {
52
53      }
54
55      def morts(l : List[Animal], n : Int) : List[Animal] = {
56
57      }
58
59  }
```