Durée du CC : 1h30.

La Fiche-Triche Scala est le seul document autorisé.

Nous allons représenter et manipuler des expressions arithmétiques sur les entiers. Les trois opérations que nous considérerons sont l'addition, la multiplication et la division.

Question 1 Déclarez une classe abstraite `Expression` (destinée à représenter nos expressions arithmétiques), et scellez-là.
Déclarez ensuite les `case class` non-mutables suivantes, qui héritent de `Expression` :

- `Entier(n)`, où `n` est un entier
- `Add(e1,e2)`, où `e1` et `e2` sont des `Expression`
- `Mult(e1,e2)` où `e1` et `e2` sont des `Expression`
- `Div(e1,e2)` où `e1` et `e2` sont des `Expression`

Question 2 Implémentez la méthode `evaluer()` de la classe `Expression`. Celle-ci renverra une `Option[Int]`.
Le résultat de `evaluer()` sera de la forme `Some(res)` s'il n'y a pas de division par zéro et que l'expression s'évalue en `res`. S'il y a une division par `0`, alors la méthode renverra `None`.

Question 3 Faites en sorte que l'égalité entre `Expression` vérifie si les deux expressions s'évaluent en la même valeur. En particulier, deux expressions non légales (i.e. qui contiennent une division par zéro) sont égales.

Question 4 Écrivez la méthode `simplifier()` de la classe `Expression`. Celle-ci renverra une nouvelle `Expression`, qui aura été simplifiée en appliquant les règles arithmétiques suivantes :

- `0` est l'élément neutre pour l'addition : autrement dit, `0+e` et `e+0` se simplifient en `e`,
- `1` est l'élément neutre pour la multiplication : `1*e` et `e*1` se simplifient en `e`,
- `0` est absorbant pour la multiplication : `0*e` et `e*0` se simplifient en `0`,
- `e/1` se simplifie en `e`

Question 5 En l'état, nos `Expression` ne peuvent pas être utilisées correctement dans des structures de données utilisant une fonction de hachage.
Rectifiez cette situation, en réimplémentant `hashCode()` de façon raisonnable.