Quand le temps d'exécution d'une fonction est long, il est tentant de *cacher* les résultats déjà calculés, pour éviter de perdre du temps la prochaine fois qu'on fera le même appel.

Question 1 Quelle est la propriété que doit posséder la fonction en question pour qu'il soit légal d'utiliser cette astuce ? Cela paraît-il raisonnable dans le monde de Scala ?

1 Version minimale de `Cache`

On va utiliser une classe `Cache` pour représenter une fonction `f:T=>U` et son cache, qui sera implémenté sous forme de `HashMap` dont les clefs/valeurs sont les `x`/`f(x)` déjà calculés. Attention : les caches que nous allons implémenter devront être **non-mutables** !
On commencera par importer `scala.collection.immutable.HashMap`

Question 2 Déclarez la classe générique `Cache`, paramétrée par les deux types `T` et `U`. Son constructeur principal sera privé (il suffit d'ajouter le mot-clef `private` entre le nom de la classe et son constructeur principal), et aura deux arguments :

- une fonction `f:T=>U`
- un `HashMap` (paramétré par les bons types).

Écrivez maintenant un constructeur public à un argument (la fonction `f`), et qui démarre avec un cache vide.

Question 3 Faites en sorte qu'on puisse appeler un `Cache c` "comme une fonction" sur un argument `x`, c'est-à-dire via `c(x)`. Cet appel devra renvoyer un couple de type `(U,Cache[T,U])`, dont le premier élément sera le résultat de `c.f` sur `x`, et le deuxième sera le nouveau `Cache`.

Il faudra évidemment, si le résultat est déjà présent dans le cache, qu'il soit utilisé sans recalcul. À l'inverse, si le résultat n'a pas encore été calculé, il devra être intégré au cache renvoyé en seconde position.

2 Test du `Cache`

On va maintenant tester la classe qu'on vient d'écrire. Pour cela, on va prendre comme exemple l'attribution fictive d'une note aux étudiants et étudiantes

Question 4 Écrire une classe `Etu` qui représente un étudiant ou une étudiante. Le constructeur acceptera uniquement un entier `numetu`, qui sera le seul attribut de la classe. Faites en sorte que l'égalité entre deux `Etu` soit déterminée par l'égalité de leur `numetu`.

Recopiez la méthode suivante, qui détermine la note d'un `Etu` en regardant les mouches voler pendant 2 secondes, puis en se basant sur le numéro d'étudiant modulo 21. [1].

```
def note(e:Etu) = {
  Thread.sleep(2000)
  e.numetu % 21
}
```

1. Toute ressemblance avec la manière dont les notes sont attribuées à la FST est purement fortuite.

Question 5 Vérifiez que votre classe `Cache` fait bien ce qu'on attend d'elle, à savoir que la deuxième fois qu'on calcule la note d'une même étudiante via un `Cache`, la réponse est donnée sans temps d'attente.

Question 6 Que se passe-t-il lorsqu'on calcule la note de `new Etu(32)`, puis qu'on calcule la note d'un autre `new Etu(32)` en utilisant le cache renvoyé par le calcul précédent ? Est-ce le comportement attendu au vu du `equals` de la classe `Etu` ?
Corrigez ce comportement.

3 Composition (☠)

Question 7 L'objectif de cette section est d'implémenter la méthode `composition` de `Cache`. Cette méthode prend comme argument un `Cache[U,V] c` (et devra donc être paramétrée par le type `V`), et renvoie le `Cache` dont la fonction est la composée de `this.f` avec `c.f`, dans le bon sens.

On souhaite que le résultat renvoyé ait un cache aussi complet que possible : si `this.f(x)` a déjà été calculé et vaut `y`, et si `c.f(y)` a déjà été calculé et vaut `z`, alors le `Cache` renvoyé devra retenir que le résultat sur `x` vaut `z`.

Pour cela, on s'aidera des deux méthodes suivantes de la classe `HashMap[T,U]` :

1. la méthode

```
partition(p: ((T,U)) => Boolean): (HashMap[T,U], HashMap[T,U])
```

   qui scinde un `HashMap[T,U]` en deux : la première composante du résultat contient toutes les paires clefs/valeurs vérifiant le prédicat `p`, et la seconde toutes celles qui ne vérifient pas `p`, et

2. la méthode

```
transform[V](g: (T,U) => V): HashMap[T,V]
```

   qui renvoie le `HashMap[T,V]` dont les clefs sont les mêmes que celui de départ, et dont les valeurs sont obtenues en appliquant la fonction `g` à ses clefs/valeurs.

On notera en particulier que l'argument de `partition` attend un couple en argument, tandis que celui de `transform` attend deux arguments.
On prendra soin de tester cette méthode, par exemple en composant le cache de `note` de la Section 2 avec un `Cache[Int,Int]` qui double les notes après quelques secondes d'attente.