Conception de logiciels intranet : intergiciels à objets répartis (NSY102)

Listes immutables

1. Des listes *simplement chaînées* peuvent être implantées sous la forme de l'interface `SimpleList<A>` et des deux types record `Nil<A>` et `Cons<A>` donnés en annexe. Vérifier le comportement des méthodes `equals`, `hashCode` et `toString`.

2. Traduire la définition ci-dessus en source Java 10 (et ajouter les méthodes `equals`, `hashCode` et `toString`). On transformera alors l'interface `SimpleList<A>` en une classe appélée `ImmutableList<A>`.

3. Ajouter les méthodes `hd` et `tl` suivantes :

   ```
   public A hd()
   ```

   ```
   public ImmutableList<A> tl()
   ```

4. Compléter l'implantation de l'interface `List<E>` de la JDK. On pourra s'appuyer sur la classe abstraite `AbstractList<E>` : il suffit alors d'implanter les méthodes `get` et `size`.

5. Pourquoi l'implantation des méthodes `hashCode` et `equals` est-elle devenue inutile ? Que dit la documentation de l'interface `List<E>` de la JDK à ce sujet ?

6. Ajouter le code nécessaire dans la classe `ImmutableList<A>` pour implanter le *pattern* visiteur. La méthode `accept` doit implanter le prototype suivant :

   ```
   public <R> R accept(ListVisitor<R, A> v)
   ```

   L'interface d'un visiteur est la suivante :

   ```
   public interface ListVisitor<R, A> {

       public R visitNil();
       public R visitCons(A h, ImmutableList<A> t);

   }
   ```

7. Ajouter des méthodes statiques `of` à `ImmutableList<A>` permettant de construire facilement des listes immutables de moins de 5 éléments.

8. Généraliser la méthode statique `of` précédente à un nombre arbitraire d'éléments :

   ```
   static <A> ImmutableList<A> list(A... elements)
   ```

Annexe

```
sealed interface SimpleList<A> {}
record Nil<A>() implements SimpleList<A> {}
record Cons<A>(A head, SimpleList<A> tail) implements SimpleList<A> {}
```

Exemple

La liste `[1;2;3;4]` est représentée ainsi :

```
new Cons<>(1, new Cons<>(2, new Cons<>(3, new Cons<>(4, new Nil<>()))))
```