

Exercices de Patterns (5)

1. L'IPL fait appel à vos services pour développer une application de streaming annexe à ecampus : SpotIPLY. Dans un premier temps, on ne désire implémenter qu'une classe Album qui garde le plus d'information possible sur les albums disponibles en écoute. Seuls le **titre** de l'album et le **nom de l'artiste** sont **obligatoires**. Les autres renseignements proposés sont le **label** de la marque de disque, le **producteur**, le **pays** de l'artiste, la **version** (ex. DeLuxe Edition, Japanese Edition, ...), le **genre**, l'**année de parution** et, **s'il s'agit d'un album remasterisé**, l'**année de parution originale**, le **débit en qualité standard** et le **débit en mode abonné**.

Comme on le voit, plusieurs solution s'offre à nous pour créer un album :

- a. Définir un unique constructeur qui accepte des valeurs null pour tous les champs sauf le titre et l'artiste : désavantage : il faut préciser des valeurs pour tous les autres champs, ne fut-ce que null.
- b. Définir une série de constructeur « télescopiques » en classant les champs dans l'ordre décroissant de fréquence d'utilisation :

```
Album(String titre, String artiste);  
Album(String titre, String artiste, int annee);  
Album(String titre, String artiste, int annee, String genre);
```

Il est en effet difficile de définir des constructeurs différents ayant le même nombre de paramètre vu que les types de ceux-ci sont trop souvent les mêmes.
 - désavantage : similaire à la solution précédente. On doit bien souvent préciser la valeur de champs qu'on ne veut pas initialiser.
- c. Utiliser le pattern JavaBeans : un constructeur sans paramètre et des getters et setters pour les autres champs
 - désavantages : il ne permet pas l'immutabilité (pas de champs final par exemple). De plus la construction se fait en différents appels. L'objet construit peut donc être dans un état inconsistant à un moment donné.

Le problème est lié au langage qui, contrairement à d'autre (Ada, Python) ne permet pas (encore) de définir des paramètres nommés optionnels.

Dans un tel cas la bonne manière de faire est de définir une classe `public static Builder` interne à la classe à construire (ici Album)

- a. La classe Album a
 - un seul constructeur `private` prenant un `Builder` en paramètre,
 - tous ses champs final.
- b. Cette classe `Builder` a
 - les mêmes champs que la classe qu'elle doit construire (ici Album),
 - un constructeur reprenant uniquement les champs obligatoires,
 - des méthodes (sorte de setters) portant le nom des autres champs mais qui renvoient le builder,
 - une méthode `public build ()` qui appelle le constructeur `private` d'Album.

Un album est donc construit par exemple comme suit :

```
Album lonerism = new Album.Builder("Lonerism",  
"Tame Impala").pays("Australie").annee(2012).genre("indie rock").build(); Album  
orange = new Album.Builder("channel ORANGE",  
"Frank Ocean").annee(2012).genre("R&B").build();  
Album visions = new Album.Builder("Visions",  
"Grimes").annee(2012).label("4AD").genre("Electronic").build();
```