SpaceFox

Zeste de savoir

Des constantes sans gâchis

9 décembre 2022

Table des matières

Introduction

«Extrayez cette valeur dupliquée *n* fois dans une constante», «Assignez cette valeur magique à une constante bien nommée et utilisez la constante à la place»… si vous avez fait du développement, vous avez sans doute reçu ces injonctions un jour ou l'autre, de la part de votre IDE, d'un collègue, d'un outil d'analyse statique de code ou autre.

Mais est-ce que ces conseils devraient s'appliquer à l'instant, de façon systématique et sans réfléchir? Évidemment, non[1]. Mais se pencher dessus va nous permettre de voir ce qu'est une constante créée de façon intelligente.

1. Une constante utile c'est une constante bien nommée

… et une constante bien nommée, c'est une constante dont le nom est **fonctionnel**. Qui indique **à quoi sert** la constante, et **ne décrit pas** son contenu. Pour savoir son contenu, il suffit d'aller voir sa valeur, merci.

C'est hélas un problème que l'on rencontre *très* souvent, soit par ignorance, soit par manque d'idée jamais corrigé.

Par exemple, ceci est une constante *mal* nommée. Elle ne nous apprends rien de plus que sa valeur et ne rends pas le code plus lisible quand elle est utilisée:

```
1 const val COLON = ":"
```

Pire, si sa valeur doit être changée, on peut se retrouver avec ce genre d'aberration.

1. Si vous avez lu les autres billets de la «série» en cours, vous avez sans doute déjà compris que la réponse à cette question est *toujours* «non». Dans le cas contraire, lisez «La «configuration» par langage dédié (DSL), une invention de Satan », «Une erreur de configuration = un plantage », «Vous n'avez (probablement) pas besoin de cette interface » et «Ne générez pas vos getters et setters! », chez le même éditeur, pas cher.

2. Ces deux valeurs identiques ont-elles besoin d'être une même constante?

```
const val COLON = ";"
```

Ce qui donne toute une catégorie de bugs *très* pénibles à détecter, parce le code ne correspond plus à ce qu'il prétend être.

Un meilleur nommage serait que la constante indique ce à quoi *sert* sa valeur:

```
const val LINUX_PATH_SEPARATOR = ":"
```

Ce qui a plusieurs avantages: on sait à quoi sert la valeur quand elle est utilisée dans le code (c'est une forme d'auto-documentation – d'ailleurs, il faudra qu'on parle de la documentation un jour…), et surtout ça permet d'éviter des réutilisations ou des modifications hors de propos. Dans ce cas, un développeur du projet sous Windows évitera de remplacer sauvagement cette valeur par `;` pour que le projet tourne chez lui, parce que la constante indique explicitement que le cas d'utilisation est Linux[2].

D'ailleurs, en parlant de réutilisation sauvage…

2. Ces deux valeurs identiques ont-elles besoin d'être une même constante ?

C'est très fréquent que les outils d'analyse de code indiquent qu'une valeur est dupliquée et devrait être factorisée dans une constante unique. Mais ces valeurs sont-elles un réel duplicata, ou est-ce une coïncidence qui *ne devrait pas* être factorisée?

Vérifiez toujours avant de regrouper des valeurs, sous peine de faire comme ce gros éditeur[3] qui avait une seule clé de traduction pour l'anglais «second», qui peut se traduire en français comme «seconde» (si on parle de l'unité temporelle») ou «deuxième» (si c'est le nombre ordinal); et donc qui affichait:

> Machine virtuelle démarrée depuis 0 deuxième(s)

Notez que les constantes bien nommées aident beaucoup à éviter ce genre de cas, parce qu'elles produisent du code absurde à la relecture:

```
// Toutes les constantes contiennent le caractère ':'
// Ça a l'air légitime, mais la constante est mal nommée
val csvEntries = csvLine.split(COLON)
```

2. On me rétorquera que le développeur capable de changer la valeur d'une constante de façon à ce qu'elle ne corresponde plus à son nom ignorera tout aussi bien les indications données par ce nom… ce qui est hélas probable, en effet. Mais si ça n'arrête pas le mauvais développeur, ça aidera tout de même à la correction du problème engendré par icelui.
3. VMware, avec vSphere

```
4
5  // On renomme la constante par ailleurs...
6  val csvEntries = csvLine.split(LINUX_PATH_SEPARATOR)
7  // Oups.
8
9  // Avec une nouvelle constante bien nommée elle aussi :
10 val csvEntries = csvLine.split(CSV_ENTRIES_SEPARATOR)
```

3. Cette valeur est-elle réellement une « valeur magique »

Certains outils d'analyse statique – et, plus gênant, certains humains – demandent à l'extraction dans une constante de toute valeur non «triviale»[4]

Mais est-ce que l'extraction dans une constante va **réellement** améliorer la lisibilité et la maintenance du code? Ça n'est pas garanti du tout.

Prenons par exemple cette classe de mapping de configuration. Les valeurs en dur dans le code sont les valeurs par défaut quand il n'y a aucune configuration particulière fournie, et ceci est un fait bien connu des développeurs qui ont accès à cette classe.

```
1  @ConfigurationProperties("fr.spacefox.example.data")
2  public class SampleDataConfiguration {
3
4      private @NotNull Duration refreshRate = Duration.ofMinutes(5);
5  }
```

Et-ce vraiment utile d'écrire ceci pour satisfaire un outil ou un collègue mal lunés?

```
1  @ConfigurationProperties("fr.spacefox.example.data")
2  public class SampleDataConfiguration {
3
4      // On est d'accord pour dire que la syntaxe de déclaration de
           constantes en Java est complètement nulle ?
5      private static final int DEFAULT_REFRESH_RATE_IN_MINUTES = 5;
6
7      private @NotNull Duration refreshRate =
           Duration.ofMinutes(DEFAULT_REFRESH_RATE_IN_MINUTES);
8  }
```

… et des exemples du même genre, il y en a *plein*.

4. En général la chaine vide et les nombres 0 et 1 sont considérés comme acceptables en dur dans le code, le reste est considéré comme devant être remplacé par «une constante bien nommée».

Et je ne parle même pas d'absurdités comme celles-ci (tirée d'exemples réels…), où la constante est à la fois mal nommée *et* moins lisible que la valeur qu'elle remplace:

```java
@ConfigurationProperties("fr.spacefox.example.data")
public class SampleDataConfiguration {

    private static final int EMPTY_STRING = "";

    private @NotNull String defaultName = EMPTY_STRING;
}
```

4. Bonus : où ranger ses constantes ?

C'est une bonne question à laquelle il n'y a pas de réponse universelle. Certains les rangent dans d'immenses classes de constantes, d'autres les éparpillent au besoin; certains ne jurent que par les `interface` de constantes, d'autres par les `class` de constantes, etc.

La seule vraie règle est: soyez cohérents. N'utilisez qu'un seul système au sein du même projet.

Conclusion

Les constantes, dans l'ensemble, c'est bien, ça apporte plein de facilités, de sûreté et d'auto-documentation au code, et même parfois un peu de performances (même si ça ne devrait pas être le but premier).

Vous devriez apporter un soin tout particulier au nommage de vos constantes, sans quoi elles deviendront des boulets plutôt que des aides. De plus, ça n'est pas parce qu'on vous dit «mettez une constante ici» qu'il faut le faire sans réfléchir. Peut-être que ça n'est pas utile. Peut-être que ça n'est pas la constante que vous pensiez réutiliser en premier lieu.
