SpaceFox

Zeste de savoir

Vous n'avez (probablement) pas besoin de cette interface

3 décembre 2022

Table des matières

Introduction

Dans certains langages de programmation[1], on a beaucoup tendance à créer des interfaces à la pelle sans trop réfléchir, puis à leur créer une et une seule implémentation. Ça fait qu'on se retrouve avec un code qui contient énormément d'interfaces `InterfaceName` et d'implémentations nommées `InterfaceNameImpl`.

Et c'est un problème.

Pourquoi c'est un problème?

— Parce que c'est signe d'un code qui n'est pas conçu, mais qui est développé en appliquant sans réfléchir des «bonnes pratiques» dont on ne sait ni d'où elles sortent, ni si elles sont applicables au cas développé.
— Parce que ça complexifie la navigation dans le code.
— Parce que ça n'apporte souvent rien d'avoir extrait cette interface: ni une meilleure lisibilité, ni une facilité de maintenance, ni une facilité d'évolution *dans les faits*[2].
— Parce que ça peut même *complexifier* la maintenance en ajoutant des modifications non-nécessaires sur l'interface *et* sur l'implémentation si l'interface doit changer, contre une seule modification s'il n'y a pas cette interface.
— Parce que ça éloigne la documentation (souvent écrite dans l'interface) de son implémentation (écrite dans la classe).

Ainsi donc:

Vous qui développez, ne créez des interfaces dans votre code que si elles sont **nécessaires** à son bon fonctionnement.

Et là on me demandera:

Dis-moi, renard, comment sais-je qu'une interface est nécessaire à mon code?

En vérité c'est assez simple. Deux cas vous indiquent que vous **devriez** avoir une interface:

1. Oui, Java, c'est à toi que je pense !
2. Je parle de faits et uniquement de faits ici, pas de théorie vue en cours ou dans des bouquins.

1. Vous développez une bibliothèque : mettez toute l'API publique dans des interfaces. Ça permet une excellente séparation de ce qui est public (et utilisable par les tiers) de ce qui est du détail d'implémentation et «public» pour raisons techniques. Vous pouvez même distribuer ces interfaces dans un paquet séparé si c'est pertinent dans votre cas.
2. Vous savez déjà, lors de votre conception, que vous avez plusieurs implémentations distinctes à réaliser – maintenant ou dans un futur proche et non-hypothétique.

Un autre cas vous indique que vous ne **devriez pas** créer d'interface: si vous vous apprêtez à en créer une seule implémentation, *surtout* si cette implémentation est tellement peu spécifique que vous ne lui avez pas trouvé meilleur nom que celui de l'interface suivi de `Impl`.

Le reste est à gérer au cas par cas, mais n'oubliez pas que les IDE modernes permettent très simplement d'extraire une interface du code existant.

Quant à l'argument très souvent lu qui prétend que *«programmer par interfaces permet très facilement de changer une implémentation, de remplacer une ancienne implémentation par une nouvelle sans changer le reste du code ou bien d'adapter le code aux nouveaux besoin»*, je voudrais vous livrer une expérience personnelle. Ça fait maintenant plus de douze ans que je développe en Java de façon continue et professionnelle, et j'ai travaillé sur des projets variés, du gros projet e-commerce au petit site de présentation de produits, en passant par des applications Android de toutes tailles, un moteur de résolution de règles, divers progiciels métiers avec quantité de règles absconses, des projets dont le premier code a vu le jour entre 2000 et 2022…

Dans tout ça, savez-vous combien de fois j'ai pu appliquer ledit argument, c'est-à-dire remplacer une implémentation par une autre sans avoir besoin de réaliser des changements massifs d'interface ?

Absolument jamais.

Cette règle est un reliquat d'anciennes pratiques de développement[3] sur de gros projets monolithiques. Elle n'a plus de raison d'être aujourd'hui.

Conclusion

Merci de m'avoir lu.

La prochaine fois, on parlera des accesseurs et mutateurs, et de cette engeance de Satan que sont les *getters/setters autogénérés*.

---

3. Quand un ingénieur écrivait des interfaces et laissait les implémentations à des batteries de techniciens. Ça a existé dans de très gros projets de très grosses entreprises, genre IBM. Les projets de taille juste inférieure gérés par des entreprises un peu moins ultra-rigoureuses ont plutôt fini en tas de spaghettis – ce qui est *largement pire* qu'un surplus d'interfaces, soit dit en passant.