M3105 – Retour vers le Fizz Buzz !
Un Fizz Buzz SOLIDe qui (en)chaîne des responsabilités

Il vous est conseillé de faire ce TD en pair-programming 😉

Souvenez-vous du **Fizz Buzz**, votre premier programme en TDD, où vous deviez implémenter le fameux jeu des cours de récréations en respectant les règles suivantes :

→ Un nombre est **Fizz** s'il est multiple de **3**
→ Un nombre est **Buzz** s'il est multiple de **5**
→ Un nombre est **FizzBuzz** s'il est multiple de **3** et multiple de **5**

A l'époque, l'approche en TDD couplée à vos premiers pas en conception avait dû vous guider vers un programme simple similaire au suivant.

```
public class FizzBuzz {


	public String donnerLaReponsePour(Integer nombre) {

		if (isFizzBuzz(nombre))
			return "fizzbuzz";

		if (isBuzz(nombre))
			return "buzz";

		if (isFizz(nombre))
			return "fizz";

		return String.valueOf(nombre);
	}


	private boolean isFizzBuzz(Integer nombre) {
		return 0 == nombre % (3 * 5);
	}


	private boolean isBuzz(Integer nombre) {
		return 0 == nombre % 5;
	}


	private boolean isFizz(Integer nombre) {
		return 0 == nombre % 3;
	}


}
```

Le tout couvert par un fichier de tests **`FizzBuzzTest`** (que vous (re)découvrirez quand vous récupérerez le code).

*Les deux premières questions doivent être traitées en **mode déconnecté** 😉*

1. Aujourd'hui, on vous demande d'étendre ce programme en ajoutant une nouvelle règle :
   → Un nombre est **bang** s'il est multiple de **7**

   Quelles instructions ajouteriez-vous pour permettre, le plus rapidement possible, au code précédent de prendre en compte cette nouvelle règle ?

2. Et si juste, après on vous demande de prendre en compte les deux règles suivantes :
   → Un nombre est **fizzbang** s'il est multiple de **3** et multiple de **7**
   → Un nombre est **buzzbang** s'il est multiple de **5** et multiple de **7**
   → Un nombre est **fizzbuzzbang** s'il est multiple de **3** et de **5** et de **7**

   Que feriez-vous ?
   Ne sentez-vous pas comme une *mauvaise odeur* dans ce code s'il doit être étendu? Quelle est-elle ?

Donc avant de procéder à une quelconque extension de ce code,
il est temps de le nettoyer pour le préparer à recevoir facilement cette extension,
*Vous pouvez maintenant passer en **mode connecté** 😉*

3. Dans votre IDE préféré, créez un projet **`fizzbuzz`** dans lequel vous importerez les fichiers sources précédents disponible dans le répertoire **`ressources/ fizzbuzz`** du dépôt https://github.com/iblasquez/enseignement-iut-m3105-conception-avancee

   Vérifiez que les tests passent AU VERT !!!
   Versionnez votre projet.

   Refactorez votre code avec un pattern de chaine de responsabilité pour le rendre plus SOLIDe (un petit diagramme de classes avant de passer au code pourrait certainement vous aider 😉)

   N'oubliez pas commiter régulièrement !!

   Ne procédez pas à l'ajout de la règle `Bang` pour le moment, concentrez-vous uniquement sur le refactoring !!!

Remarque : Le programme présenté sur la première page couvrait les besoins demandés (**fizz**, **buzz** et **fizzbuzz**) avec une implémentation simple du problème.
Il n'était pas nécessaire à ce moment de mettre en place une conception avancée (respect des principes *KISS* et *YAGNI*, évitez *l'over-engineering* 😉)
Mais dès lors que ce problème doit être étendu et que la complexité du code augmente, il est maintenant nécessaire de réorganiser l'architecture de ce code pour ***procéder plus facilement aux nouvelles extensions*** et rendre ce code ***facilement maintenable*** par la suite.