CY TECH	**ING2 – Computer Systems Engineering** **Test et Vérification - TDD**	
	Subject : **Informatique**	*Date* : **Novembre 2023**
		Duration :
		Number of pages : **red3page.3**

Les katas suivants sont extraits du site Coding Dojo.

1 Fizz Buzz

Exercise 1.

Imaginez la scène. Vous avez onze ans et, dans les cinq minutes qui précèdent la fin du cours, votre professeur de mathématiques décide de rendre son cours plus "amusant" en introduisant un "jeu". Il explique qu'il va montrer du doigt chaque élève à tour de rôle et leur demander de dire le prochain chiffre dans l'ordre, en commençant par un. La partie "amusante" est que si le nombre est divisible par trois, vous dites "Fizz" et s'il est divisible par cinq, vous dites "Buzz". Maintenant, votre professeur de mathématiques pointe du doigt tous vos camarades de classe à tour de rôle, et ils crient joyeusement "un !", "deux !", "Fizz !", "quatre !", "Buzz !"... jusqu'à ce qu'il vous pointe très délibérément du doigt, vous fixant d'un regard d'acier... le temps s'arrête, votre bouche s'assèche, vos paumes deviennent de plus en plus moites jusqu'à ce que vous réussissiez enfin à dire "Fizz !". Le malheur est évité, et le doigt pointé passe à autre chose.

Bien entendu, pour éviter de vous mettre dans l'embarras devant toute votre classe, vous devez faire imprimer la liste complète pour savoir quoi dire. Votre classe compte environ 33 élèves et il se peut qu'il fasse trois fois le tour avant que la cloche ne sonne pour la récréation. Le prochain cours de mathématiques a lieu jeudi. Faites du code !

Écrivez un programme qui affiche les nombres de 1 à 100. Mais pour les multiples de trois, affichez "Fizz" à la place du nombre et pour les multiples de cinq, imprimez "Buzz". Pour les nombres qui sont des multiples de trois et de cinq, affichez "FizzBuzz".

Exemple de sortie :

```
1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz
16
17
Fizz
19
Buzz
... etc jusqu'  100
```

a. Vous allez réaliser ce Kata en utilisant la méthode TDD. Un étudiant écrit un test simple et son binôme. L'objectif n'est pas de coder directement la résolution, mais d'écrire un test le plus simple possible et pour le binôme, d'écrire le code le plus simple pour faire passer le test.

b. Étape 2 : On ajoute les deux règles suivantes :

   - Un nombre est fizz s'il est divisible par 3 ou s'il possède un 3
   - Un nombre est buzz s'il est divisible par 5 ou s'il possède un 5

   Modifiez vos tests et votre code.

2 Calculateur de chaîne

Exercise 2. L'objectif est de créer une fonction `add` qui prend en paramètre une chaîne et retourne une chaîne.

```
String add(String number)
```

N'anticipez pas chaque question, allez au bout avant de passer à la suivante. Chaque question ne doit pas invalider les précédentes.

a. Story 1 : Écrire la fonction `add` respectant les contraintes suivantes

   - la chaîne est constituée de 0, 1, 2 ou 3 nombres, séparés par des virgules et retourne leur somme,
   - une chîne vide retourne 0,
   - exemples de chaînes : "", "1", "1.1,2.2",
   - une erreur de formation de la chaîne retourne une exception IllegalArgumentException.

b. Story 2 : la chaîne est constituée d'un nombre indéterminé de valeurs.

c. Story 3 : le séparateur de valeurs peut accepter le caractère '\n'.

   - "1\n2,3" retourne 6,
   - "175.2,\n35" est invalide et retourne une exception `IllegalArgumentException` contenant le message "Number expected but '\n' found at position 6.".

d. Story 4 : gérer le manque d'un nombre en dernière position.

   - "1,2," retourne une exception `IllegalArgumentException` contenant "Missing number, EOF found"

e. Story 5 : gérer un séparateur propre. Le séparaeur est défini en début de chaîne de la façon suivante :

   $$//[\text{separateur}]\backslash n[\text{nombres}]$$

   - "//;\n1;2" doit retourner "3"
   - "//|\n1|2|3" doit retourner "6"
   - "//sep\n2sep3" doit retourner "5"
   - "//|\n1|2,3" est invalide et doit retourner le message suivant : "'|' expected but ',' found at position 3."

f. Story 6 : Les nombres négatifs. La chaîne ne doit pas contenir de nombre négatif. Si tel est le cas, la fonction retournera une exception `IllegalArgumentException` contenant le message "Negative not allowed : " suivi de la liste des nombres négatifs présents dans la chaîne

   - "-1,2" est invalide et doit retourner une exception `IllegalArgumentException` contenant le message "Negative not allowed : -1"

- "2,-4,-5" est invalide et doit retourner une exception `IllegalArgumentException` contenant le message "Negative not allowed : -4, -5"

g. Story 7 : Erreurs multiples. Un paramètre contenant plusieurs erreurs conduit à une exception `IllegalArgumentException` contenant un message d'erreur listant toutes les erreurs présentes.
Exemple : "-1„2" donne "Negative not allowed : -1expected but ',' found at position 3."

3 Chiffres romains

Exercise 3. Les Romains étaient des gens intelligents. Ils ont conquis la majeure partie de l'Europe et l'ont gouvernée pendant des centaines d'années. Ils ont inventé le béton, les routes droites et même les Mais il y a une chose qu'ils n'ont jamais découverte : le zéro !

Cela a rendu un peu plus difficile l'écriture et la datation des histoires détaillées de leurs exploits, mais le système de chiffres qu'ils ont inventé est toujours utilisé aujourd'hui.

Les Romains écrivaient les nombres à l'aide des lettres : I, V, X, L, C, D, M. (ces lettres ont beaucoup de lignes droites et sont faciles à graver sur des tablettes de pierre).

a. Story 1 : Écrire une fonction qui convertit des nombres entiers en chiffres romains. Ce n'est pas la peine de tester les nombres de plus de 3000.

```
1 –> I
10 –> X
7 –> VII
```

b. Story 2 : Écrire une fonction qui réalise la transformation inverse.