**TD2 – Test fonctionnel et exceptions (Jean-Marie Mottu)**

# Première partie – Conception de tests fonctionnels

Nous travaillons sur le cas d'étude d'une classe d'OperationsBinaires pour faire des calculs sur des **couples** de nombre. Avec une approche fonctionnelle on n'exploite que la spécification pour concevoir les tests. Ici considérons toutes les informations ci-dessous :

- la première ligne de la javadoc spécifie les « exigences »
- les @ de la javadoc complété par la signature des méthodes spécifieny le domaine d'entrée.

```
package but1.iut.r203

/**
 * @author mottu-jm
 */
class OperationsBinaires {
    …

    /**
     * Additionner deux entiers
     * @param op1, op2 : int Opérandes à additionner
     * @return somme : int
     * @throws ArithmeticException : out of Int bounds
     */
    fun additionner(op1: Int, op2: Int): Int {        …    }

    /**
     * Soustraire deux entiers
     * @param op1, op2 : int Opérandes à soustraire
     * @return resultat des soustractions
     * @throws ArithmeticException : out of Int bounds
     */
    fun soustraire(op1: Int, op2: Int): Int {    …    }

    /**
     * Multiplier deux entiers
     * @param op1, op2 : int Opérandes à multiplier
     * @return produit : int
     * @throws ArithmeticException : out of Int bounds
     */
    fun multiplier(op1: Int, op2: Int): Int {         …    }

    /**
     * Diviser deux entiers naturels
     * @param dividende : entier naturel
     * @param diviseur : entier naturel
     * @return quotient : flottant
     * @throws ArithmeticException
     */
    fun diviserNaturel(dividende: Int, diviseur: Int): Float {         …    }

    /**
     * Calcul de la factorielle d'un entier n positif ou nul
     * @param int n un nombre dont on veut calculer la factorielle
     * @return le résultat n! = 1*2*...*n et 0! = 1
     * @throws IllegalArgumentException
     *          quand on essaie une factorielle d'un nombre négatif
     * @throws ArithmeticException : out of Int bounds
     */
    fun factorielle(n: Int): Int {            …    }
    …
}
```

Pour effectuer le test fonctionnel par Analyse Partitionnelle, on réalise plusieurs étapes :

1. Identifiez les **variables** qui forment chaque Donnée de Test, ainsi que les variables qui seront contrôlées par l'Oracle.
2. Réalisez pour chacune des variables formant les Données de Test une **analyse partitionnelle**, afin d'en déduire des classes d'équivalences.
    a. identifiez le **type** de la variable
    b. identifiez la **plage** de la variable
        i. Des intervalles **nominaux**
        ii. La/les plages de valeurs du fonctionnement **exceptionnel**
    c. identifiez des partitions **fonctionnelles**
3. Etablissez une **table de décision** décrivant le comportement attendu.
4. Déduisez un ensemble fini de **Cas de Test**.

**Question 2.1 :** Concevoir des tests pour les différentes méthodes de la classe : (1h20)

a. `diviserNaturel`
b. `factorielle`
c. `additionner`
d. `soustraire`
e. `multiplier`

## Deuxième partie – Programmation des cas de test

Récupérez le code dans ce dépôt gitlab :

https://univ-nantes.io/iut.info1.qd1.automatisationtests/butinfo1-qd1-td2

**Question 2.2 :** Programmez les cas de tests obtenus à la précédente question. En particulier, il est nécessaire d'implémenter des tests vérifiant la bonne levée d'exception.

**Question 2.3 :** De nombreux tests échouant, corrigez le code des méthodes qui ne considèrent pas les comportements exceptionnels.

## Troisième partie – Tests aux limites

**Question 2.4 :** Reprenez les tests conçus et programmez pour `diviserNaturel` et `factorielle`, étendez-les pour considérer les tests aux limites.