Activté 2 C306 - Ingénierie du logiciel

Session 2021 - Semestre 2

Consortium International e-Miage - Auteur: *Sébastien Choplin* sebastien.choplin@u-picardie.fr

Le devoir sera rendu sous la forme d'une archive (zip) contenant:

- un fichier pdf synthétisant le travail (copier-coller des codes et remarques éventuelles)
- les différents codes développés et rapports générés.

Exercice 1 : Couverture de Code

Classe 'Calcul'

```java
public class Calcul {
  /** Calcul la somme de deux nombres **/
  public static int somme(int a,int b){
    return a+b;
  }
  public static int maFonction(int a, int b) {
   if (b >= 10) {
    return a/b;
   }
   return b;
  }
  /**
    * @return a / b si b != 0
    * @throw IllegalArgumentException si b == 0
    */
  public static int division(int a,int b){
   if ( b == 0 ) {
     throw new IllegalArgumentException("b ne doit pas etre 0");
   }
   return a / b;
  }
}
```

Tests unitaires pour la classe 'Calcul'

```java
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;

public class CalculTest {
 @Test
 public void testConstructeur() {
  new Calcul();
 }
 @Test
 public void testSomme() {
  assertEquals(5,Calcul.somme(2,3));
 }
 @Test
 public void testDivision() {
  assertEquals(4,Calcul.division(8,2));
 }
}
```

1. Réécrire le code pour qu'il passe les vérifications de *checkstyle*, *spotbugs* et *PMD*.
2. Donner le rapport de couverture de code des tests unitaires de 'CalculTest' sur la classe 'Calcul'.
3. Expliquer pourquoi la couverture n'est pas complète (en couverture de ligne et en couverture de branche).
4. Proposer une amélioration des tests pour obtenir une couverture complète (donner le résultat de couverture obtenu).

Exercice 2 : Compilation assistée

1. Organiser l'arborescence du code et des tests telle que prévue par l'outil 'maven'
2. Construire un fichier "pom.xml" pour 'maven' permettant de
    - compiler le code
    - compiler les tests (en utilisant *JUnit5*)
    - exécuter les tests unitaires
    - générer les rapports de *checkstyle*, *spotbugs* et *PMD*
    - générer le rapport de couverture du code sur les tests unitaires (en utilisant *JaCoCo*)
    - générer la documentation javadoc
    - générer les différents rapports au format HTML
3. Donner la commande permettant d'exécuter toutes les lignes de la question 2 en une fois.
4. Donner les rapports HTML générés
5. Déposer les données utiles (pom.xml, src) **dans le dossier "activite2"** du dépôt (SVN ou GIT) utilisé précédemment.

Exercice 3 : Sudoku

Le but de l'exercice est de développer une implémentation d'une grille de Sudoku pour préparer l'activité 3. Le code sera organisé suivant l'**arborescence pour 'maven' qui sera utilisé pour compiler, exécuter les tests unitaires et générer les rapports**.

Voici l'interface "Grille" permettant de modéliser une grille de Sudoku (de 4x4, 9x9, 16x16 ou 25x25):

```
public interface Grille {
    /** Caractere de case vide */
    char EMPTY = '@';

    /**
     * Caractere possible a mettre dans la grille
     * pour une grille 4x4 : 1..4
     * pour une grille 9x9 : 1..9
     * pour une grille 16x16: 0..9-a..f
     * pour une grille 25x25: 0..9-a..o
     */
    char[] possible = new char[] {
       '1', '2', '3', '4', '5', '6', '7', '8', '9', '0',
       'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
       'k', 'l', 'm', 'n', 'o'
       };

    /** @return largeur/hauteur de la grille */
    int getDimension();

    /**
     * Affecte une valeur dans la grille
     * @param x       position x dans la grille
     * @param y       position y dans la grille
     * @param value   valeur a mettre dans la case
     * @throw HorsBornesException si x ou y sont hors bornes (0-8)
     * @throw ValeurImpossibleException si la valeur est interdite aux vues des autres valeurs de la grille
     * @throw CaractereInterditException si value n'est pas un caractere autorise ('1',...,'9')
     */
    void setValue(int x, int y, char value) throws HorsBornesException, ValeurImpossibleException,
  CaractereInterditException;

    /**
     * Recupere une valeur de la grille
     * @param x      position x dans la grille
     * @param y      position y dans la grille
     * @return valeur dans la case x,y
     * @throw HorsBornesException si x ou y sont hors bornes (0-8)
     */
    char getValue(int x, int y) throws HorsBornesException;

    /**
     * Test si une grille est terminee
     * @return true si la grille est complete
     */
    boolean complete();

    /**
     * Test si une valeur est possible dans la grille par rapport a ce qu'elle
     * contient deja
     * @param x     position x dans la grille
     * @param y     position y dans la grille
     * @param value valeur a mettre dans la case
     * @throw HorsBornesException si x ou y sont hors bornes (0-8)
     * @throw CaractereInterditException si value n'est pas un caractere autorise ('1', ..., '9', ...)
     */
    boolean possible(int x, int y, char value) throws HorsBornesException, CaractereInterditException;
}
```

1. Ecrire les classes
   - *HorsBornesException* (exception levée lorsque les paramètres de positions sortent de la grille),
   - *ValeurImpossibleException*,
   - *CaractereInterditException*
2. Ecrire les tests
3. Ecrire une implémentation dans une classe "GrilleImpl".

```
/** Implementation d'une grille */
public class GrilleImpl implements Grille {
   ...
}
```

3. Donner le résultat des tests unitaires, la couverture de code sur l'exécution de ces tests unitaires, les rapports *checkstyle*, *spotbugs* et *PMD*.