Fabrice Moore le 4 septembre 2022

Master Emiage UCB Lyon 1

Module C306

Table des matières

[**Exercice 1 : Couverture de code** 2](#_Toc113120230)

[**1. Réécrire le code pour qu’il passe les vérifications de checkstyle, spotbugs et PMD.** 2](#_Toc113120231)

[**2. Donner le rapport de couverture de code des tests unitaires de ‘CalculTest’ sur la classe ‘Calcul’.** 2](#_Toc113120232)

[**3. Expliquer pourquoi la couverture n’est pas complète (en couverture de ligne et en couverture de branche).** 4](#_Toc113120233)

[**4. Proposer une amélioration des tests pour obtenir une couverture complète (donner le résultat de couverture obtenu).** 4](#_Toc113120234)

[**Exercice 2 : Compilation assistée** 5](#_Toc113120235)

[**1. Organiser l’arborescence du code et des tests telle que prévue par l’outil ‘maven’** 5](#_Toc113120236)

[**2. Construire un fichier “pom.xml” pour ‘maven’ permettant de compiler le code** 5](#_Toc113120237)

[**3. Donner la commande permettant d’exécuter toutes les lignes de la question 2 en une fois.** 8](#_Toc113120238)

[**4. Donner les rapports HTML générés** 8](#_Toc113120239)

[**5. Déposer les données utiles (pom.xml, src) dans le dossier “activite2” du dépôt (SVN ou GIT) utilisé précédemment.** 8](#_Toc113120240)

[**Exercice 3 : Sudoku** 8](#_Toc113120241)

[**1.** **Ecrire les classes** 8](#_Toc113120242)

[**2.** **Ecrire les tests** 8](#_Toc113120243)

[**3.** **Ecrire une implémentation dans une classe “GrilleImpl”** 8](#_Toc113120244)

**Exercice 1 : Couverture de code**

**1. Réécrire le code pour qu’il passe les vérifications de checkstyle, spotbugs et PMD.**

Les vérifications de Checkstyle/Spotbugs et PMD font ressortir 130 alertes (Warnings) et pas d’erreur de code. Le code compile et marche mais la mise en forme du code laisse à désirer.

Il s’agit principalement de faiblesses dans :

* L’indentation
* La documentation du code
* La dénomination des variables
* Le caractère final des variables et des méthodes etc.

Le code correct se trouve ci-dessous :

**package** Calcul;

/\*\*

\*An abstract class that support .

\*

\* **@author** Fabrice Moore

\*

\* **@version** 1.0

\*/

**public** **class** Calcul {

//\*\*

/\* Calcule la somme de deux entiers

\*\*/

**public** **static** **int** somme(**final** **int** entier1, **final** **int** entier2) { //\*somme algébrique

**return** entier1 + entier2;

}

/\*\*

\* **@return** entier1 / entier2 si entier2 != 0

\* **@throw** IllegalArgumentException si entier2 == 0

\*/

**public** **static** **int** maFonction(**final** **int** entier1, **final** **int** entier2) {

**if** (entier2 >= 10) {

**return** entier1 / entier2;

}

**return** entier2;

}

/\*

\* @return a / b si b != 0.

\* @throw IllegalArgumentException si b == 0

\*/

**public** **static** **int** division(**final** **int** entier1, **final** **int** entier2) {

**if** (entier2== 0) {

**throw** **new** IllegalArgumentException("b ne doit pas etre 0");

}

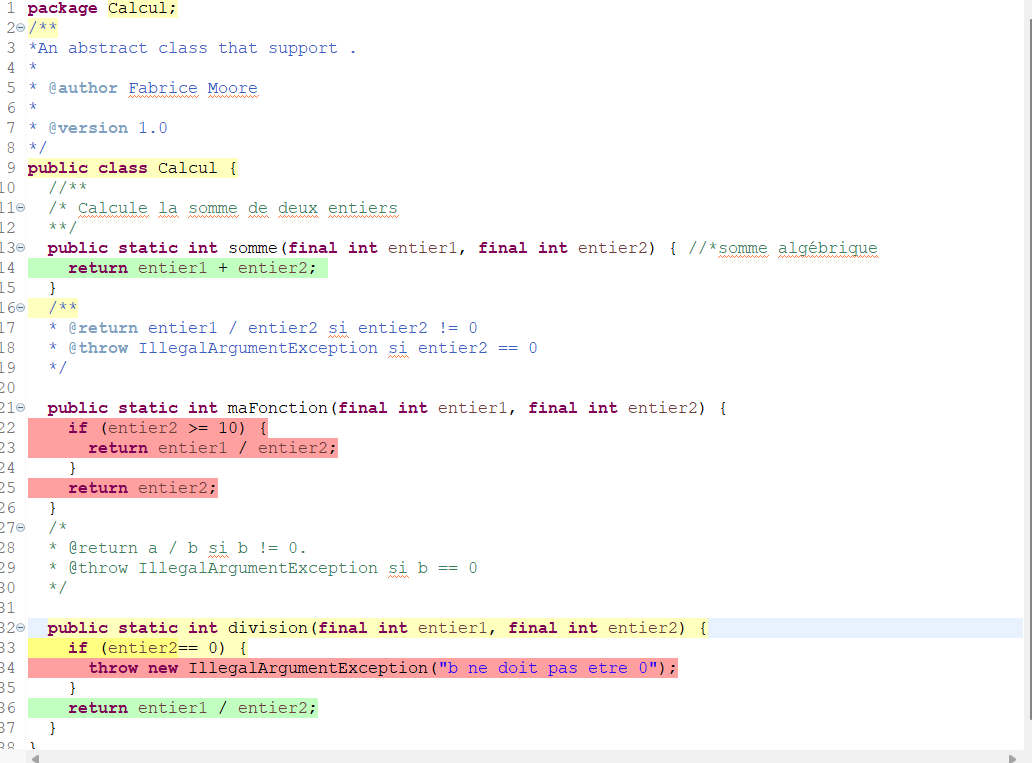
**return** entier1 / entier2;

}

}

**2. Donner le rapport de couverture de code des tests unitaires de ‘CalculTest’ sur la classe ‘Calcul’.**

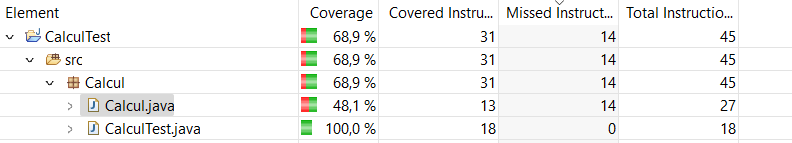
**Le rapport de la classe se présente ci-dessous**

****

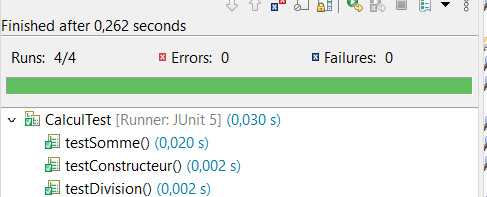
On remarque en particulier que :

* Les opérations en elles-mêmes (somme et division) sont testées puisqu’elles sont marquées vertes
* La méthode MaFonction n’est pas testée puisqu’il est en rouge
* Les conditions (if) ne sont pas testés puisqu’elles sont en rouge
* L’exception de la méthode division n’est pas testée puisqu’elle est en rouge
* En conséquence certaines méthodes sont marquées en jaune puisque le test est partiel

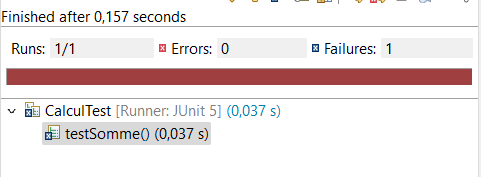
La couverture des tests est de 48.1% comme le montrent les éléments ci-dessous

****

**Si la couverture des tests est partielle, on reconnait toutefois que le résultat des tests est positif**

****

**Le changement des valeurs des tests génère bien des erreurs. Si on met 6 dans le résultat attendu pour le testSomme avec les mêmes valeurs entrée, on a bien un message d’erreur**

****

**3. Expliquer pourquoi la couverture n’est pas complète (en couverture de ligne et en couverture de branche).**

La couverture en ligne et la couverture en branche ne sont pas complètes car il manque à la fois des lignes qui ne sont pas testées et des transitions qui ne le sont pas non plues

Il convient d’insérer

* Le test du constructeur
* Des tests de la fonction maFonction
* Le test de l’exception pour le diviseur

**4. Proposer une amélioration des tests pour obtenir une couverture complète (donner le résultat de couverture obtenu).**

**Pour le test du constructeur**

**public** **class** CalculTest {

@Rule

**final** ExpectedException thrown = ExpectedException.~~none~~();

@Test

**public** **void** testConstructeur() {

**new** Calcul();

}

**Pour le test de MaFonction**

@Test

**public** **void** testMafonction\_entier2\_PlusGrand10() {

*assertEquals*(2, Calcul.*maFonction*(24, 12));

}

@Test

**public** **void** testMafonction\_entier2\_PlusPetit10() {

*assertEquals*(3, Calcul.*maFonction*(6, 3));

}

**Pour le test des exceptions en cas de dénominateur à 0**

@Test

**public** **void** manageException() {

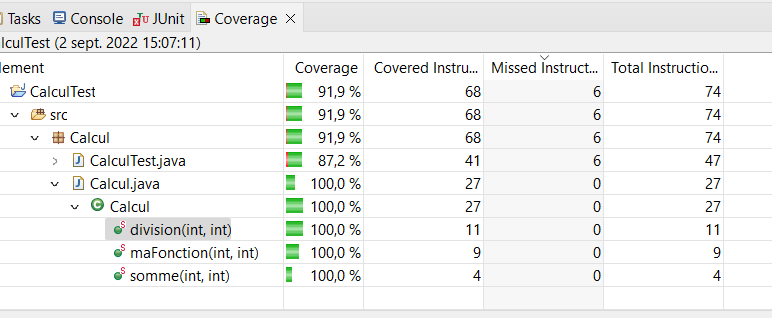
thrown.expect(IllegalArgumentException.**class**);

thrown.expectMessage("b ne doit pas etre 0");

Assertions.*assertEquals*(2, Calcul.*division*(8, 0));

}

**On constate que la couverture augmente immédiatement à 100%**

****

**Exercice 2 : Compilation assistée**

**1. Organiser l’arborescence du code et des tests telle que prévue par l’outil ‘maven’**

Je l’ai fait dans Eclipse : une chose me surprend toutefois :

Pour que cela marche il me faut deux fichiers CalculTest.java

Le premier se dans le répertoire source

Le second dans le répertoire de test (src/test/java)

**2. Construire un fichier “pom.xml” pour ‘maven’ permettant de compiler le code**

Voici le code utilisé dans le fichier POM

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <name>CalculTest</name>

  <description>UCB</description>

  <version>1.1</version>

  <groupId>default</groupId>

  <artifactId>CalculTest</artifactId>

  <dependencies><!-- dependendance du projet -->

    <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter -->

<dependency>

        <groupId>org.junit.jupiter</groupId>

        <artifactId>junit-jupiter-engine</artifactId>

        <version>5.3.1</version>

        <scope>test</scope>

    </dependency>

  </dependencies>

  <build>

    <plugins>

    <plugin>

    <groupId>org.apache.maven.plugins</groupId>

    <artifactId>maven-compiler-plugin</artifactId>

    <version>3.8.0</version>

    <configuration>

    <release>7</release>

    </configuration>

      </plugin>

      <plugin>

      <artifactId>maven-project-info-reports-plugin</artifactId>

          <version>3.0.0</version>

      </plugin>

      <plugin><!-- pour l'execution des tests -->

      <groupId>org.apache.maven.plugins</groupId>

      <artifactId>maven-surefire-plugin</artifactId>

      <version>3.0.0-M3</version>

      <configuration>

<testFailureIgnore>true</testFailureIgnore>

</configuration>

      </plugin>

      <plugin><!-- pour la generation de rapports -->

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-site-plugin</artifactId>

        <version>3.7.1</version>

        <configuration>

          <locales>fr,en</locales><!-- en francais par defaut -->

        </configuration>

      </plugin>

      <plugin><!-- pour la couverture de code avec JaCoco lors de l'exécution des tests JUnit -->

      <groupId>org.jacoco</groupId>

      <artifactId>jacoco-maven-plugin</artifactId>

      <version>0.8.4</version>

      <executions>

          <execution>

            <goals>

              <goal>prepare-agent</goal>

            </goals>

          </execution>

          <execution>

            <id>report</id>

            <phase>prepare-package</phase>

            <goals>

              <goal>report</goal>

            </goals>

          </execution>

      </executions>

      </plugin>

      <plugin><!-- pour l'analyse avec checkstyle -->

      <groupId>org.apache.maven.plugins</groupId>

      <artifactId>maven-checkstyle-plugin</artifactId>

      <version>3.0.0</version>

      </plugin>

    </plugins>

  </build>

  <reporting><!-- on indique ici les rapports qu'on veut obtenir lors de la phase de generation de rapports -->

    <plugins>

      <plugin>

        <groupId>org.jacoco</groupId>

        <artifactId>jacoco-maven-plugin</artifactId>

        <reportSets>

          <reportSet>

            <reports>

              <report>report</report>

            </reports>

          </reportSet>

        </reportSets>

      </plugin>

      <plugin><!-- pour avoir le rapport checkstyle -->

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-checkstyle-plugin</artifactId>

        <version>3.0.0</version>

        <reportSets>

          <reportSet>

            <reports>

              <report>checkstyle</report>

            </reports>

          </reportSet>

        </reportSets>

      </plugin>

      <plugin><!-- pour avoir le lien au code source dans les rapports -->

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-jxr-plugin</artifactId>

        <version>2.3</version>

      </plugin>

      <plugin><!-- pour la verification du code avec PMD -->

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-pmd-plugin</artifactId>

        <version>3.12.0</version>

      </plugin>

      <plugin><!-- pour avoir le rapport surfire (execution des tests) -->

        <groupId>org.apache.maven.plugins</groupId>

        <artifactId>maven-surefire-report-plugin</artifactId>

        <version>3.0.0-M3</version>

      </plugin>

    </plugins>

  </reporting>

  <properties>

    <!-- encodage du code source -->

    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

  </properties>

</project>

**3. Donner la commande permettant d’exécuter toutes les lignes de la question 2 en une fois.**

Je clique droit sur POM.xml et lance MAVEN build

**4. Donner les rapports HTML générés**

Voir fichiers zip

**5. Déposer les données utiles (pom.xml, src) dans le dossier “activite2” du dépôt (SVN ou GIT) utilisé précédemment.**

Si vous lisez ce rapport c’est que j’ai réussi à déposer le fichier dans GITHUB

**😊**

**Exercice 3 : Sudoku**

1. **Ecrire les classes**

**Pour la classe** HorsBornesException, je propose :

**public** **class** HorsBornesException {

**public** **static** **void** HorsBornesException(**int** ligne, y colonne) **throws** Exception {

**if** (ligne>8||colonne>8)

**throw** **new** IllegalArgumentException("x ou y sont hors bornes (0-8)");

}

}

**Pour la classe** CaractereInterditException, je propose :

**public** **class** CaractereInterditException {

**public** **static** **void** CaractereInterditException(**char** value) **throws** Exception {

**if** (!String.*valueOf*(possible).contains(value)) {

**throw** **new** IllegalArgumentException("Seules les valeurs autorisées dans le tableau sont acceptées");

}

}

}

**Pour la classe** ValeurImpossibleException, **je propose de ne pas créer d’exception, mais plutôt d’utiliser un booléen nécessaire au bon déroulement du programme. IL faut reconnaitre qu’il ne s’agit pas d’une exception en soit mais plutôt d’un test pour accepter un nombre qui fait partie du bon déroulement du programme :**

**public** **class** ValeurImpossibleException {

**public** **static** **boolean** CaractereInterditException(**boolean** correct) {

**if** (!correct) {

System.***out***.println("Cette valeur n'est pas possible");

}

**return** **true**;

}

}

1. **Ecrire les tests**

**Test de** HorsBornesException

@Test

**public** **void** testException() {

thrown.expect(IllegalArgumentException.**class**);

thrown.expectMessage("x ou y sont hors bornes (0-8)");

Assertions.*assertEquals*(2, HorsBornesException(18, -1));

}

}

**Test de** CaractereInterditException

@Test

**public** **void** manageException() {

thrown.expect(IllegalArgumentException.**class**);

thrown.expectMessage("Seules les valeurs autorisées dans le tableau sont acceptées");

Assertions.*assertEquals*(2, CaractereInterditException(18, -1));

}

}

**Test de ValeurImpossibleException**

@Test

**public** **void** testGetReponse() {

**boolean** correct = **false**;

String expResult = "Cette valeur n'est pas autorisée";

String result = ValeurImpossibleException(correct);

*assertEquals*(expResult, result);

}

}

1. **Ecrire une implémentation dans une classe “GrilleImpl”**

**Voici une proposition**

**public** **class** GrilleImpl **implements** Grille{

**static** **int** [][]*grille* = {

{0,0,0,0,3,0,0,6,2},

{0,0,0,0,7,2,0,0,1},

{2,0,0,6,0,0,8,0,0},

{1,0,9,4,0,6,0,8,7},

{0,0,4,0,0,0,2,0,0},

{5,8,0,1,0,7,6,0,4},

{0,0,6,0,0,1,0,0,3},

{4,0,0,3,8,0,0,0,0},

{7,3,0,0,6,0,0,0,0}

};

**public** **static** **boolean** estDansLaColonne(**int** valeur,**int** colonne) {

**for** (**int** ligne=0; ligne<9; ligne++) {

**if** (*grille*[ligne][colonne]==valeur) **return** **true**;

}

**return** **false**;

}

**public** **static** **boolean** estDansLaLigne(**int** valeur,**int** ligne) {

**for** (**int** colonne=0; colonne<9; colonne++) {

**if** (*grille*[ligne][colonne]==valeur) **return** **true**;

}

**return** **false**;

}

**public** **static** **boolean** estDansLeCarre(**int** valeur,**int** ligne,**int** colonne) {

**int** pointGauche = 3\*(colonne/3);

**int** pointHaut = 3\*(ligne/3);

**for** (**int** c=pointGauche; c<pointGauche+3; c++) {

**for** (**int** l=pointHaut; l<pointHaut+3; l++) {

**if** (*grille*[l][c]==valeur) **return** **true**;

}

}

**return** **false**;

}

**public** **static** **int** *compteur*;

**public** **static** **boolean** estValeurPossible(**int** valeur,**int** ligne,**int** colonne) {

**return** !*estDansLaColonne*(valeur,colonne)

&& !*estDansLaLigne*(valeur,ligne)

&& !*estDansLeCarre*(valeur,ligne,colonne);/\*estDansLeCarre(valeur,ligne,colonne);\*/

}

**public** **static** **void** affiche() {

**for** (**int** l=0; l<9; l++) {

**for** (**int** c=0; c<9; c++) {

**if** (*grille*[l][c]!=0) System.***out***.print(*grille*[l][c]);

**else** System.***out***.print('-');

}

System.***out***.println();

}

System.***out***.println();

}

**public** **static** **boolean** trouveSolution(**int** ligne,**int** colonne) {

*compteur*++;

// calcul de la position suivante

**int** ligneSuivante;

**int** colonneSuivante;

**if** (colonne==8) { ligneSuivante = ligne+1; colonneSuivante=0; }

**else** { ligneSuivante = ligne; colonneSuivante = colonne+1; }

// Est-ce que j'ai parcouru toutes les cases du tableau ?

**if** (ligne==9) { *affiche*(); **return** **true**; }

**if** (*grille*[ligne][colonne]!=0) { // case déjà remplie ==> case suivante

**return** *trouveSolution*(ligneSuivante,colonneSuivante);

} **else** { // case non remplie ==> je dois essayer les différentes valeurs

**for** (**int** valeur=1; valeur<10; valeur++) {

// si on ne peut pas poser la valeur, on passe à la suivante

**if** (!*estValeurPossible*(valeur,ligne,colonne)) **continue**;

*grille*[ligne][colonne] = valeur; // je pose une valeur (hypothèse)

**boolean** correct = *trouveSolution*(ligneSuivante,colonneSuivante);

**if** (correct) **return** **true**; // c'est bon, la valeur était correcte

}

*grille*[ligne][colonne] = 0; // c'était une case "vide", on la remet

**return** **false**; // Je n'ai pas trouvé de valeur correcte

}

}

**public** **static** **void** main(String []a) {

*affiche*();

*compteur* = 0;

System.***out***.println(*trouveSolution*(0,0)+" nombre d'appels="+*compteur*);

}

}

**4 Donner les rapports PMD,Checkstyle,SpotBugs**

**Les rapports seront remis après le prochain contact du 7 septembre**