ACTIVITE N°2

Exercice 1

1-Reecriture des codes

//Classe Calcul

package mypackage;

/\*\*

\* Calcul.

\* @author GBIKPI BENISSANH Date E.

\*/

public final class Calcul {

/\*\*

\* Contructeur privee.

\*/

private Calcul() {

}

/\*\*

\* Calcul la somme de deux nombres.

\* @param a premiere valeur entier

\* @param b second valeur entier

\* @return la somme de a et b

\*/

public static int somme(final int a, final int b) {

return a + b;

}

/\*\*

\* @param a

\* @param b

\* @return la divion de a par b si b superieur ou egal à 10 sinon retourne b

\*/

public static int maFonction(final int a, final int b) {

final int nbre = 10;

if (b >= nbre) {

return a / b;

}

return b;

}

/\*\*

\* @param a

\* @param b

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

\* @throw IllegalArgumentException si b == 0

\*/

public static int division(final int a, final int b) {

if (b == 0) {

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

}

return a / b;

}

}

package mypackage;

import static org.junit.jupiter.api.Assertions.assertEquals; // import pour la méthode statique assertEquals

import static org.junit.jupiter.api.Assertions.assertThrows; //import pour la méthode statique assertThrows

import org.junit.jupiter.api.Test; // import pour l'annotation @Test indiquant qu'il s'agit d'une méthode de test

public class CalculTest {

/\*\*

\* Vérifie le constructeur de la classe.

\*/

@Test

public void testConstructeur() {

//

}

/\*\*

\* Vérifie la somme de a et b.

\*/

@Test

public void testSomme() {

final int a = 2;

final int b = 3;

final int expected = 5;

assertEquals(expected, Calcul.somme(a, b));

}

/\*\*

\* Vérifie la division de a et b.

\*/

@Test

public void testDivision() {

final int a = 8;

final int b = 2;

final int expected = 4;

assertEquals(expected, Calcul.division(a, b));

}

/\*\*

\* mafontion avec b superieur à 10. return la division de a/b

\*/

@Test

public void testMaFonctionAvecBSuperieurADix() {

final int a = 20;

final int b = 10;

final int expected = 2;

assertEquals(expected, Calcul.maFonction(a, b));

}

/\*\*

\* mafontion avec b inferieur à 10. retrun b

\*/

@Test

public void testMaFonctionAvecBInferieurADix() {

final int a = 10;

final int b = 5;

final int expected = 5;

assertEquals(expected, Calcul.maFonction(a, b));

}

/\*\*

\* Division par 0. return une exception capturé

\*/

@Test

public void testDivisionParZero() {

final int a = 4;

final int b = 0;

Throwable exception = assertThrows(IllegalArgumentException.class,

() -> {

Calcul.division(a, b);

});

assertEquals("b ne doit pas etre 0", exception.getMessage());

}

}

2- Le rapport de couverture de code et test unitaire se trouve dans le dossier «exo1/rapport ».

Note : les dossiers before contiennent les rapports générés avant et les dossiers after contiennent les rapports après corrections des remarques générés

3- La couverture de code n’est pas complète car toutes les méthodes de la classe Calcul ne sont pas testées.

4-Amelioration du code. Le rapport de couverture se trouve dans le dossier « exo1/rapport/couverture\_code/after\_rapport »

Exercice 2

1. Code organisé suivant l’architecture de maven
2. Fichier pom.xml se trouve dans le dossier « exo2/pom.xml »
3. Commande permettant d’exécuter en une seule commande les lignes de la question 2 :

Mvn clean compile test site :site

1. Les rapports générés se trouve dans le dossier « exo2/site »

Exercice 3

1. Ecriture des classes

* HorsBornesException

/\*\*

\* HorsBornesException.

\* @author GBIKPI BENISSANH Date E.

\*/

public class HorsBornesException extends Exception {

/\*\*

\* Constructeur de la classe HorsBornesException.

\* @param errorMessage message d'erreur

\*/

public HorsBornesException(final String errorMessage) {

super(errorMessage);

}

}

* CaractereInterditException

/\*\*

\* CaractereInterditException.

\* @author GBIKPI BENISSANH Date E.

\*/

public class CaractereInterditException extends Exception {

/\*\*

\* Constructeur de la classe CaractereInterditException.

\* @param errorMessage message d'erreur

\*/

public CaractereInterditException(final String errorMessage) {

super(errorMessage);

}

}

* ValeurImpossibleException

/\*\*

\* ValeurImpossibleException.

\* @author GBIKPI BENISSANH Date E.

\*/

public class ValeurImpossibleException extends Exception {

/\*\*

\* Constructeur de la classe ValeurImpossibleException.

\* @param errorMessage message d'erreur

\*/

public ValeurImpossibleException(final String errorMessage) {

super(errorMessage);

}

}

1. Ecriture des tests

* HorsBornesExceptionTest

import static org.junit.jupiter.api.Assertions.\*; //import des static methods

import org.junit.jupiter.api.Test;

/\*\*

\* HorsBornesExceptionTest.

\* @author GBIKPI BENISSANH Date E.

\*/

class HorsBornesExceptionTest

{

/\*\*

\* Constructeur de la classe HorsBornesException.

\*/

@Test

public void testConstructeur()

{

final String ExpectedMessage = "Les positions x ou y sont hors bornes";

final String ErrorMessage = "Les positions x ou y sont hors bornes";

HorsBornesException ex = new HorsBornesException(ErrorMessage);

assertEquals(ExpectedMessage, ex.getMessage());

}

}

* CaractereInterditExceptionTest

import static org.junit.jupiter.api.Assertions.\*; //import des static methods

import org.junit.jupiter.api.Test;

/\*\*

\* CaractereInterditExceptionTest.

\* @author GBIKPI BENISSANH Date E.

\*/

class CaractereInterditExceptionTest

{

/\*\*

\* Constructeur de la classe CaractereInterditException.

\*/

@Test

public void testConstructeur()

{

final String ExpectedMessage = "value n'est pas un caractere autorise";

final String ErrorMessage = "value n'est pas un caractere autorise";

CaractereInterditException ex = new CaractereInterditException(ErrorMessage);

assertEquals(ExpectedMessage, ex.getMessage());

}

}

* ValeurImpossibleExceptionTest

import static org.junit.jupiter.api.Assertions.\*; //import des static methods

import org.junit.jupiter.api.Test;

/\*\*

\* ValeurImpossibleExceptionTest.

\* @author GBIKPI BENISSANH Date E.

\*/

class ValeurImpossibleExceptionTest

{

/\*\*

\* Constructeur de la classe ValeurImpossibleException.

\*/

@Test

public void testConstructeur()

{

final String ExpectedMessage = "Valeur interdite!";

final String ErrorMessage = "Valeur interdite!";

ValeurImpossibleException ex = new ValeurImpossibleException(ErrorMessage);

assertEquals(ExpectedMessage, ex.getMessage());

}

}

1. Implémentation de la classe GrilleImpl

/\*\*

\* GrilleImpl.

\* @author GBIKPI BENISSANH Date E.

\*/

public class GrilleImpl implements Grille {

/\*\*

\* tableau des valeurs.

\*/

private final char[][] magrille;

/\*\*

\* dimension de la grille.

\*/

private final int dimension;

/\*\*

\* borne maximale.

\*/

private final int bornemax = 8;

/\*\*

\* Constructeur de la classe GrilleImpl.

\* @param x est un entier designat la taille de la grille à creer

\*/

public GrilleImpl(final int x) {

this.dimension = x;

this.magrille = new char[x][x];

initGrille();

}

/\*\*

\* retourne la dimension du la grille.

\* @return la dimension

\*/

public int getDimension() {

return this.dimension;

}

/\*\*

\* affecte une valeur à la cellule definie.

\* @param x ligne de la grille

\* @param y colonne de la grille

\* @param value valeur de la cellule

\* @throws HorsBornesException

\* @throws CaractereInterditException

\* @throws ValeurImpossibleException

\*/

public void setValue(final int x, final int y, final char value)

throws HorsBornesException,

CaractereInterditException, ValeurImpossibleException {

if (x > bornemax || y > bornemax) {

throw

new HorsBornesException("Les positions x "

+ "ou y sont hors bornes");

}

if (!checkValue(value)) {

throw

new CaractereInterditException(value + " n'est pas "

+ "un caractere autorise");

}

boolean ligneOk = verifieLigne(x, value);

boolean colonneOk = verifieColonne(y, value);

if (ligneOk && colonneOk) {

throw

new ValeurImpossibleException(value + " est "

+ "une valeur interdite");

}

this.magrille[x][y] = value;

}

/\*\*

\* retourne la valeur de la cellule.

\* @param x ligne de la grille

\* @param y colonne de la grille

\* @return la valeur de la cellule

\* @throws HorsBornesException

\*/

public char getValue(final int x, final int y) throws HorsBornesException {

if (x > bornemax || y > bornemax) {

throw

new HorsBornesException("Les positions x ou "

+ "y sont hors bornes");

}

return this.magrille[x][y];

}

/\*\*

\* @return true si pas de vide sinon false.

\*/

public boolean complete() {

for (int i = 0; i < this.dimension; i++) {

for (int j = 0; j < this.dimension; j++) {

if (this.magrille[i][j] == EMPTY) {

return false;

}

}

}

return true;

}

/\*\*

\* Test si une valeur est possible dans la grille par rapport a ce qu'elle

\* contient deja.

\* @param x ligne de la grille

\* @param y colonne de la grille

\* @param value valeur à inserer

\* @return true si la valeur est possible sinon false

\* @throws CaractereInterditException

\* @throws HorsBornesException

\*/

public boolean possible(final int x, final int y, final char value)

throws CaractereInterditException,

HorsBornesException {

if (x > bornemax || y > bornemax) {

throw

new HorsBornesException("Les positions x "

+ "ou y sont hors bornes");

}

if (!checkValue(value)) {

throw

new CaractereInterditException(value + " n'est "

+ "pas un caractere autorise");

}

boolean ligneOk = verifieLigne(x, value);

boolean colonneOk = verifieColonne(y, value);

return ligneOk && colonneOk;

}

/\*\*

\* initialise la grille.

\*/

private void initGrille() {

//

for (int i = 0; i < this.dimension; i++) {

for (int j = 0; j < this.dimension; j++) {

this.magrille[i][j] = EMPTY;

}

}

}

/\*\*

\* verifie les valeurs sur la colonne.

\* @param y colonne definie

\* @param value valeur à rechercher

\* @return un bolean

\*/

private boolean verifieColonne(final int y, final char value) {

boolean colonneOk = true;

for (int i = 0; i < this.dimension; i++) {

//

if (this.magrille[i][y] == value) {

colonneOk = false;

break;

}

}

return colonneOk;

}

/\*\*

\* verifie la valeur sur la ligne.

\* @param x valeur de la ligne

\* @param value valeur à comparer

\* @return un bolean

\*/

private boolean verifieLigne(final int x, final char value) {

boolean ligneOk = true;

for (int j = 0; j < this.dimension; j++) {

if (this.magrille[x][j] == value) {

ligneOk = false;

break;

}

}

return ligneOk;

}

/\*\*

\* retourne true si la valeur existe dans Possible sinon false.

\* @param value valeur à rechercher

\* @return un bolleen

\*/

private boolean checkValue(final char value) {

boolean found = false;

for (char c : POSSIBLE) {

if (c == value) {

found = true;

break;

}

}

return found;

}

}

1. Rapport de test unitaires et couverture de code se trouve dans le dossier « exo3/site »

NB :

L’activité se trouve sur le dépôt github à l’adresse ci-dessous :

Activité 2

<https://github.com/clarkdev700/Activite2.git>

Activité 1

<https://github.com/clarkdev700/Activite1.git>