ACTIVITE N°1

Exercice 1

1-Le code ne compile pas parce qu'il contient des erreurs:

- La méthode « setPrix » tente de modifier une valeur déclarée constante
- Il manque un return lorsque le prix n'est pas positif
- Il manque le mot clé « this » pour référencer le champ « reference » dans le constructeur du Produit

2- Correction de code

Déclaration Initial	Correction
private final double prix;	private double prix;

public final String getReference() {
String resultat = reference;
if (prix > 0) return reference;
}
	public final String getReference() {
	if (prix > 0) {
	return reference;
	}
	return null;
	}

public Produit(String reference) {reference = reference;}	public Produit(final String lreference) {
	this.reference = lreference;
	}

3- Les rapports se trouvent dans le dossier « Exercice1/before »

- rapport-checkstyle.txt
- rapport-pmd.txt
- rapport-spotbugs.html

4- Ecriture du code

Confère fichier « Produit.java » dans le dossier Exercice 1

```
package mypackage;


import java.util.Objects;
/**
 * Classe Produit représentant un produit avec un prix et une référence.
 */
public class Produit {
    /** Défini la référence du produit . */
    private final String reference;
    /** Défini le prix du produit . */
    private double prix;
```

```
/**
 * Crée un produit avec la référence fourni.
 * @param lreference
 */
public Produit(final String lreference) {
    this.reference = lreference;
}

/**
 * Retourne le prix d'un produit.
 * @return le prix
 */
public double getPrix() {
    return this.prix;
}

/**
 * modifie le prix.
 * @param lprix
 */
public final void setPrix(final Double lprix) {
    this.prix = lprix;
}

/**
 * @return la reference si le prix est positif, null sinon
 */
public final String getReference() {
    if (prix > 0) {
        return reference;
    }
```

```
        return null;
    }

    /**
     * Compare deux instances de produit.
     * @param o type Object
     * @return vrai si l'objet à la même réference et faux si non
     */
    @Override
    public boolean equals(final Object o) {
        if (o == null || !(o instanceof Produit)) {
            return false;
        }
        return reference == ((Produit) o).reference;
    }

    /**
     * Redefinir la methode hashCode.
     * @return le hascode de la reference
     */
    @Override
    public int hashCode() {
        return Objects.hashCode(reference);
    }
}
```

5- Les rapports appliqués au code réécrit se trouvent dans le dossier « Exercice1/after »

- rapport-checkstyle.txt
- rapport-pmd.txt
- rapport-spotbugs.html

Exercice 2

1- Ecriture des tests unitaire avec JUnit 5

Confère fichier « TestTabAlgos.java » dans le dossier Exercice 2

// TestTabAlgos.java

```
package mypackage;

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

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

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

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


/**
 *
 * @author E. Date GBIKPI BENISSANH
 */
@SuppressWarnings("PMD.EmptyCatchBlock")
public final class TestTabAlgos {


    /**
     * Détermine la valeur la plus grande d'un tableau.
     */
    @Test
    public void testplusGrand() {
        final int[] tab = new int[]{5, 52, 99, 8, 20, 87, 23};
        final int expectedValue = 99;
        assertEquals(expectedValue, TabAlgosUtils.plusGrand(tab));
    }
    /**
     * Calcule la moyenne des valeurs du tableau.
     */
    @Test
```

```
public void testMoyenne() {
    final int[] tab1 = new int[]{5, 52, 99, 8, 20, 87, 23};
    final int expectedValue = 42;
    assertEquals(expectedValue, TabAlgosUtils.moyenne(tab1));
}

/**
 * Calcule la moyenne des valeurs du tableau en levant une exception.
 *
 */
@Test
public void testMoyenneAvecException() {
    int[] tab2 = null;
    try {
        assertEquals(0, TabAlgosUtils.moyenne(tab2));
        fail("Exception levé pour le tableau sans paramètre.");
    } catch (IllegalArgumentException e) {
        // This is expected
    }
}

/**
 * Compare le contenu de 2 tableaux en tenant compte de l'ordre.
 *
 */
@Test
public void testEgaux() {
    final int[] tab3 = new int[]{5, 2, 7, 8, 10, 24, 23};
    final int[] tab4 = new int[]{5, 2, 7, 8, 10, 24, 23};
    assertTrue(TabAlgosUtils.egaux(tab3, tab4));
}
```

```
    /**
     * Compare le contenu de 2 tableaux sans tenir compte de l'ordre.
     *
     */
    @Test
    public void testSimilaires() {
        final int[] tab5 = new int[]{5, 2, 7, 8, 10, 24, 23};
        final int[] tab6 = new int[]{10, 24, 23, 5, 8, 2, 7};
        assertTrue(TabAlgosUtils.similaires(tab5, tab6));
    }
}
```

2- Implémentation des codes

Confère fichier « TabAlgosUtils.java »

```
package mypackage;


public final class TabAlgosUtils {


    /**
     * Constructeur protected avec exception pour empecher l'instantiation de la
     * classe.
     *
     * @throws java.lang.Exception
     */
    private TabAlgosUtils() throws Exception {
        throw new Exception("cette classe ne peut pas être instanciée");
    }


    /**
     * @return valeur la plus grande d'un tableau.
     * @param tab
     */
```

```
public static int plusGrand(final int[] tab) {
    int x = tab[0];
    for (int i = 1; i < tab.length; i++) {
        if (tab[i] > x) {
            x = tab[i];
        }
    }
    return x;
}

/**
 * @return moyenne des valeurs du tableau.
 * @throw IllegalArgumentException si tab et null ou vide.
 * @param tab
 */
public static double moyenne(final int[] tab) {
    int total = 0;
    if (tab == null || tab.length == 0) {
        throw new IllegalArgumentException("le tableau doit "
                + "contenir des valeurs");
    }

    for (int i : tab) {
        total += i;
    }
    return total / (double) tab.length;
}

/**
 * Compare le contenu de 2 tableaux en tenant compte de l'ordre.
 *
 * @return true si les 2 tableaux contiennent les mêmes éléments avec les
```

```
 * mêmes nombres d'occurences (avec les elements dans le meme ordre).
 * @param tab1
 * @param tab2
 */
public static boolean egaux(final int[] tab1, final int[] tab2) {
    if  (tab1.length != tab2.length) {
        return false;
    }

    for (int i = 0; i < tab1.length; i++) {
        if (tab1[i] != tab2[i]) {
            return false;
        }
    }
    return true;
}

/**
 * Compare le contenu de 2 tableaux sans tenir compte de l'ordre.
 *
 * @return true si les 2 tableaux contiennent les mêmes éléments avec les
 * mêmes nombres d'occurrence (pas forcément dans le même ordre).
 * @param tab1
 * @param tab2
 */
public static boolean similaires(final int[] tab1, final int[] tab2) {
    if  (tab1.length != tab2.length) {
        return false;
    }
```

```
        int tabLength = tab1.length;
        boolean flag;
        for (int i = 0; i < tabLength; i++) {
            int x = tab1[i];
            flag = false;
            for (int j = 0; j < tabLength; j++) {
                if (x == tab2[j]) {
                    flag = true;
                    break;
                }
            }

            if (!flag) {
                return false;
            }
        }
        return true;
    }
}
```

3- Vérification de la validité du code

4- Les différents rapports se trouvent dans les dossiers « Exercice2/before » (rapport avant correction) et « Exercice2/after » (rapport après correction).

Chaque dossier contient les rapports suivant :

- rapport-checkstyle.txt
- rapport-pmd.txt
- rapport-spotbugs.html
- rapport-checkstyle-test.txt
- rapport-pmd-test.txt
- rapport-spotbugs-test.html

NB : Le rapport du test unitaire se trouve dans le fichier :

« rapport-junit.txt »

Les codes java compilé se trouvent dans le dossier « Exercice2/out/mypackage »