Exercice1

1.Le code ne compile pas parce qu'il a 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 this pour référencer le champ reference dans le constructeur du produit Produit.

2.Correction du code

```
private double prix;
public final String getReference() {
 if (prix > 0) {
 return reference;
}
return null;
 }

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

Le rapport applique au code reecrit se trouve dans

Excercice 2

1.Ecriture des test unitaire avec JUnit5

```
package mypackage;

import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
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
```

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


@SuppressWarnings("unused")
class TabAlgosTest {

	/**
	 * 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.Implementation des codes

```
package mypackage;

import static org.junit.jupiter.api.Assertions.*;


import org.junit.jupiter.api.Test;

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;
               }


}
```