

Numéro d'inscription : 1080 HF

Nom : RAVOLAMPY

Prénoms : Victoire Lydia

Niveau : M1 IG

TP GENIE LOGICIEL

## TP JUnit :

- 1- Création de la classe 'Money' ainsi que la classe 'MoneyTest'.

```
public class Money {
    private int fAmount;
    private String fCurrency;

    public Money(int amount, String currency) {
        fAmount = amount;
        fCurrency = currency;
    }

    public int amount() {
        return fAmount;
    }

    public String currency() {
        return fCurrency;
    }

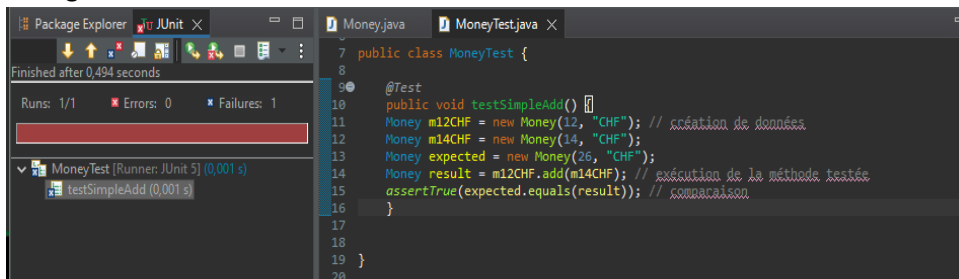
    public Money add(Money m) {
        return new Money(fAmount() + m.amount(), fCurrency());
    }
}
```

- 2- Ajout de la méthode de test à la classe 'MoneyTest', qui vérifie l'addition de deux montants avec la même devise.

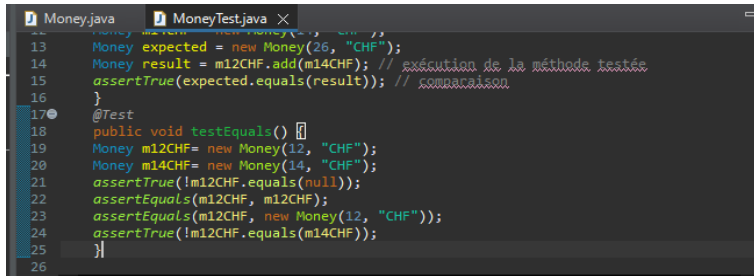
```
public class MoneyTest {

    @Test
    public void testSimpleAdd() {
        Money m12CHF = new Money(12, "CHF"); // création de données
        Money m14CHF = new Money(14, "CHF");
        Money expected = new Money(26, "CHF");
        Money result = m12CHF.add(m14CHF); // exécution de la méthode testée
        assertTrue(expected.equals(result)); // comparaison
    }
}
```

- 3- La méthode de test a été ajoutée à la classe MoneyTest. Ensuite, lorsqu'on exécute le test de la méthode 'add' en utilisant : « Run, Run as, JUnit test », le test a échoué, car la méthode equals n'est pas encore implémenter dans notre classe. Comme nous pouvons constater sur l'image ci-contre.

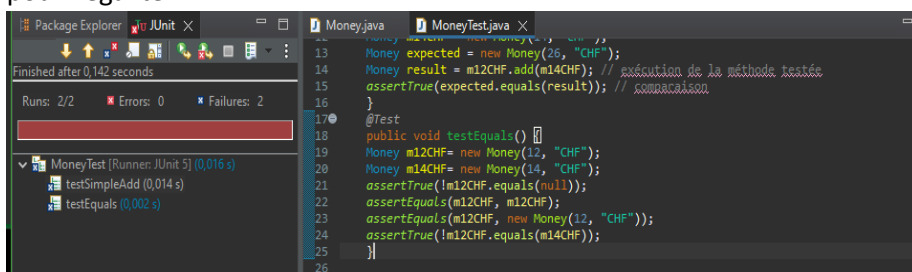


- 4- Ajout de la méthode de test d'égalité 'TestEquals' dans la classe de test 'MoneyTest'.



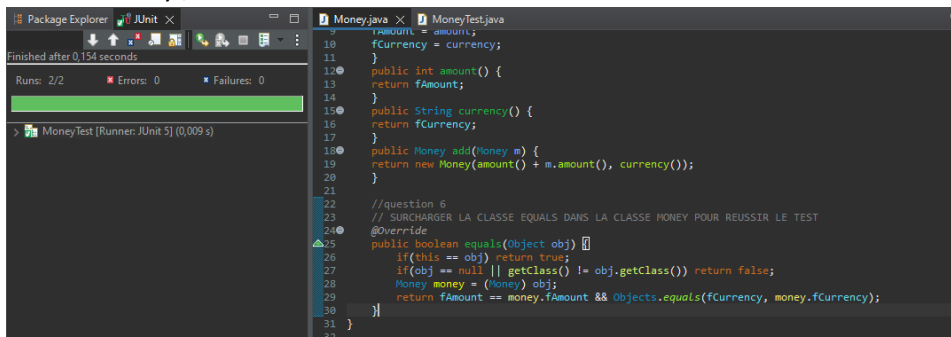
```
13 Money expected = new Money(26, "CHF");
14 Money result = m12CHF.add(m14CHF); // exécution de la méthode testée
15 assertTrue(expected.equals(result)); // comparaison
16 }
17
18 @Test
19 public void testEquals() {
20     Money m12CHF = new Money(12, "CHF");
21     Money m14CHF = new Money(14, "CHF");
22     assertTrue(!m12CHF.equals(null));
23     assertEquals(m12CHF, m12CHF);
24     assertEquals(m12CHF, new Money(12, "CHF"));
25     assertTrue(!m12CHF.equals(m14CHF));
26 }
```

- 5- Après avoir relancer la classe 'MoneyTest', on constate qu'à part la première insertion, le test a été échoué pour toutes les assertions parce que la méthode 'equals' n'est pas encore surchargée dans la classe 'Money'. Les instances de 'Money' ne sont donc pas comparées pour l'égalité.



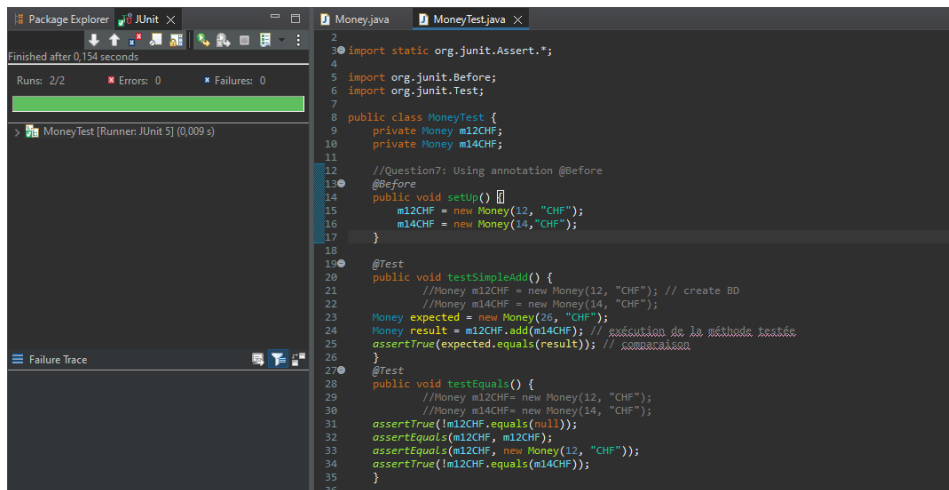
```
13 Money expected = new Money(26, "CHF");
14 Money result = m12CHF.add(m14CHF); // exécution de la méthode testée
15 assertTrue(expected.equals(result)); // comparaison
16 }
17
18 @Test
19 public void testEquals() {
20     Money m12CHF = new Money(12, "CHF");
21     Money m14CHF = new Money(14, "CHF");
22     assertTrue(!m12CHF.equals(null));
23     assertEquals(m12CHF, m12CHF);
24     assertEquals(m12CHF, new Money(12, "CHF"));
25     assertTrue(!m12CHF.equals(m14CHF));
26 }
```

- 6- Ainsi, pour passer ce test avec succès, nous allons surcharger la méthode 'equals' de la classe 'Money', comme suit :



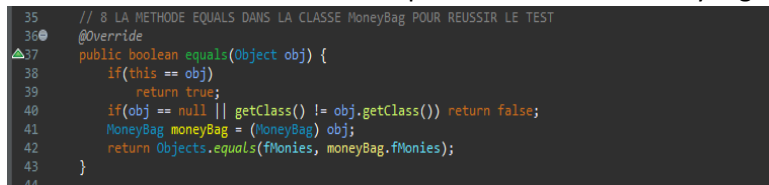
```
10 fAmount = amount;
11 fCurrency = currency;
12 }
13 public int amount() {
14     return fAmount;
15 }
16 public String currency() {
17     return fCurrency;
18 }
19 public Money add(Money m) {
20     return new Money(amount() + m.amount(), currency());
21 }
22
23 //question 6
24 // SURCHARGER LA CLASSE EQUALS DANS LA CLASSE MONEY POUR REUSSIR LE TEST
25 @Override
26 public boolean equals(Object obj) {
27     if(this == obj) return true;
28     if(obj == null || getClass() != obj.getClass()) return false;
29     Money money = (Money) obj;
30     return fAmount == money.fAmount && Objects.equals(fCurrency, money.fCurrency);
31 }
32 }
```

- 7- Pour pouvoir remédié à la duplication de code dans les méthodes 'testSimpleAdd' et 'testEquals'. Nous allons utiliser l'annotation '@Before' pour créer des objets Money communs à utiliser dans plusieurs méthodes de test. Ensuite, nous allons relancer le test pour vérifier que nos modifications n'ont pas altéré le résultat.

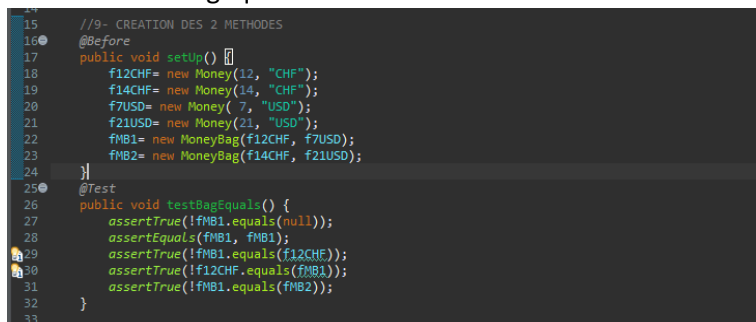


Maintenant que la classe 'Money' semble fonctionner pour une unique devise, nous allons prendre en charge des devises multiples. Nous allons ainsi introduire la classe 'MoneyBag' pour agréger des valeurs de différentes devises.

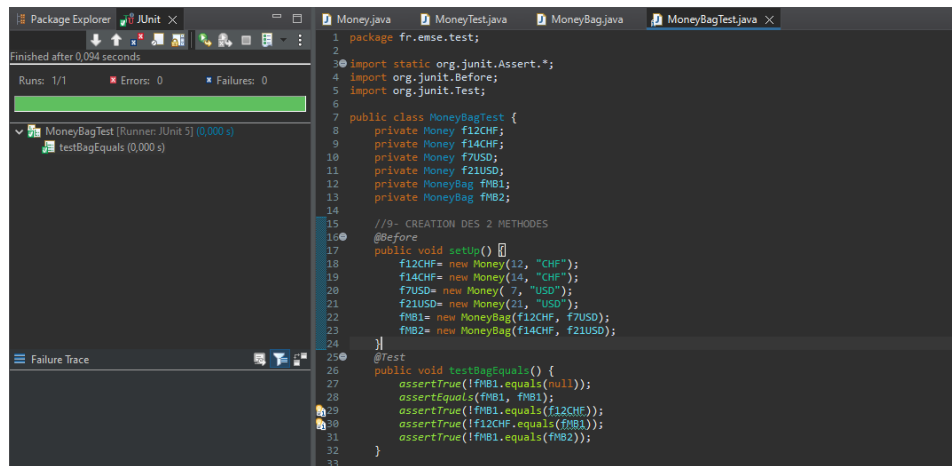
- 8- Nous allons écrire la méthode 'equals' de la classe 'MoneyBag'.



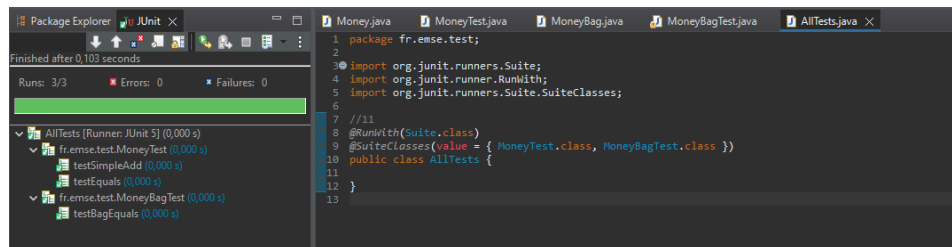
- 9- Création de la classe de test 'MoneyBagTest' avec une méthode annotée '@Before' et une méthode 'testBagEquals'.



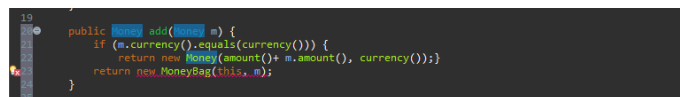
- 10- On constate alors que le test a été effectué avec succès. Et la modification nécessaire à apporter peut-être l'implémentation des méthodes de test dans la classe MoneyBagTest que nous allons voir à la question 12.



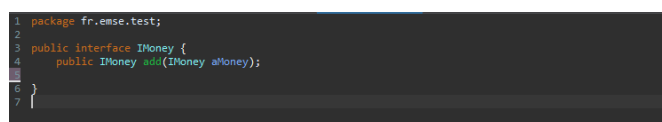
11- Création d'une nouvelle suite de test 'AllTests' afin de tester 'Money' et 'MoneyBag' (Test les uns à la suite des autres).



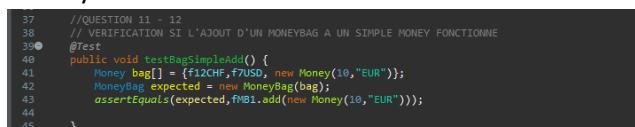
Maintenant, nous allons corriger la méthode 'add' de la classe 'Money'.



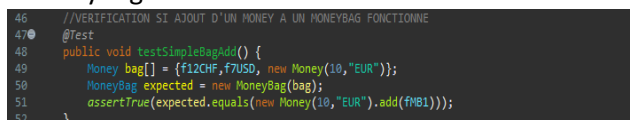
A cause d'un problème de typage, cette méthode ne va pas compiler. Introduisons alors une interface 'IMoney' que doivent implémenter Money et MoneyBag.



12- Définissons la méthode 'testBagSimpleAdd' pour ajouter une 'MoneyBag' à un simple Money.



Définissons la méthode 'testSimpleBagAdd' pour ajouter un simple Money à une 'MoneyBag'.



Définissons la méthode 'testBagBagAdd' pour ajouter deux 'MoneyBag'.

```
53 //VERIFICATION SI AJOUT DE DEUX MONEYBAG FONCTIONNE
54 @Test
55 public void testBagBagAdd() {
56     Money bag[] = {f12CHF, f7USD, f14CHF, f21USD};
57     MoneyBag expected = new MoneyBag(bag);
58     assertEquals(expected, fMB1.add(fMB2));
59 }
60
```

13- Afin que les tests unitaires soient passés, nous allons modifier 'IMoney', 'Money', ainsi que 'MoneyBag'.

Modification de la classe 'Money'.

```
29 //QUESTION 13
30
31 public IMoney add(IMoney m) {
32     return m.addMoney(this);
33 }
34
35 public IMoney addMoney(Money m) {
36     if (m.currency().equals(currency())) {
37         return new Money(amount()+m.amount(), currency());
38     }
39     return new MoneyBag(this, m);
40 }
41 @Override
42 public IMoney addMoneyBag(MoneyBag mb) {
43     return mb.addMoney(this);
44 }
45
```

Modification de la classe 'MoneyBag'.

```
44
45 //13
46 public MoneyBag(Money... m) {
47     for (Money money : m)
48         appendMoney(money);
49 }
50 @Override
51 public IMoney add(IMoney m) {
52     return m.addMoneyBag(this);
53 }
54 @Override
55 public IMoney addMoneyBag(MoneyBag mb) {
56     MoneyBag result = new MoneyBag();
57     for (Money money : this.fMonies)
58         result.appendMoney(money);
59     for (Money monet : mb.fMonies)
60         result.appendMoney(monet);
61     return result;
62 }
63 @Override
64 public IMoney addMoney(Money m) {
65     appendMoney(m);
66     if (fMonies.size() == 1) {
67         return fMonies.get(0);
68     } else {
69         return this;
70     }
71 }
72
```

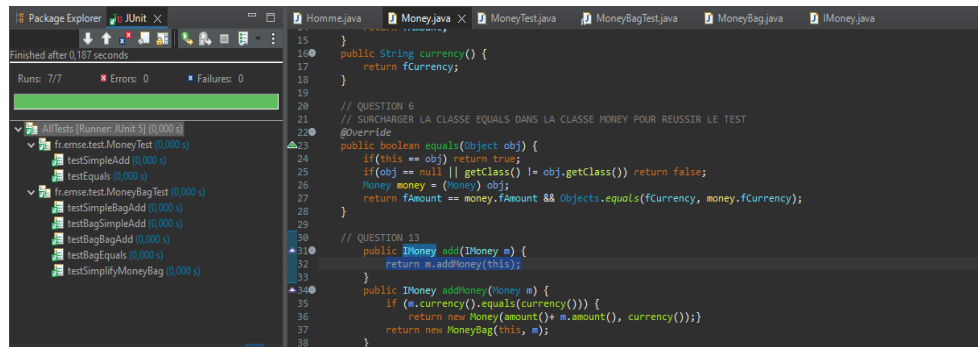
Modification de l'Interface 'IMoney'

```
1 package fr.emse.test;
2
3 public interface IMoney {
4     public IMoney add(IMoney aMoney);
5     //13
6     public IMoney addMoney(Money m);
7     public IMoney addMoneyBag(MoneyBag mb);
8 }
9
```

14- Définissons maintenant un jeu de test pour vérifier la simplification des MoneyBag en Money lorsque cela est nécessaire.

```
61 // QUESTION 14
62 //SIMPLIFICATION DES MONEYBAG EN MONEY (JEU DE TEST)
63 @Test
64 public void testSimplifyMoneyBag() {
65     //CREE UN MONEYBAG AVEC UNE SEULE VALEUR EN CHF
66     MoneyBag moneyBag = new MoneyBag(new Money(12, "CHF"));
67     IMoney result = moneyBag.add(new Money(-12, "CHF")); //AJOUTER -12 CHF AU MONEYBAG? SIMPLIFICATION MONEYBAG EN UN
68     assertTrue(result instanceof Money); //VERIFIER SI LE RESULTAT EST UN MONEY
69     assertEquals(new Money(0, "CHF"), result); // VERIFIER SI LE MONEY RESULTANT A UNE VALEUR DE 0 CHF
70 }
71
72
```

15-



## TP Refactoring

9- Il y a une erreur lors du déplacement de la méthode 'toString()' vers les sous classes.