TP cycle de vie des applications
S. Salva

TP3 Tests Unitaires

1. Objectifs

Savoir développer des tests unitaires
Effectuer la couverture de test d'une application
Utiliser Infinitest
Unitiliser Mockito

2.

L'objectif est d'implanter une classe Money ayant pour paramètres *amount* et *curr* représentant un montant et une devise respectivement.
La devise s'exprime comme une chaîne de trois caractères définie par la norme ISO (ici, seulement EUR (euro), USD ($), CHF (franc suisse), GBP (livre sterling)).

Quand on additionne deux Money de même devise, le résultat a pour montant la somme des deux autres montants. Mais si les devises sont différentes alors il faut convertir. La structure de la classe est la suivante:

```
class Money {

private int amount ;
private String curr ;


public Money( int amount , String currency );

public int amount ( ) ;

public String currency ( );

public void add (Money m);

public void add (int namount , String ncurrency );

}
```

Dans la suite , il vous est demandé de coder des tests avec assertions par contrats.

A

En partant de la structure ci-dessus, écrire une classe MoneyTest qui teste si Money est correcte (pour chaque méthode, un cas de test est demandé).

Réfléchissez bien et donnez tous les scénarios possibles afin d'élaborer tous vos tests. De même, n'oubliez pas la fixture. Nous nous limiterons aux devises EUR, USD (1 EUR = 1.29 USD au moment de l'écriture du sujet).

Ecrivez chaque cas de test, puis chaque méthode de la classe Money de façon incrémentale (l'une après l'autre, petit à petit donc en TDD). En fin de code, la classe Money
implantant l'en-tête donnée doit également passer tous vos tests.

Si vous trouvez barbant de relancer vos tests à la main, vous pouvez installer Infinitest (Install new software et donner l'URL *http://infinitest.github.io*

Infinitest recherche les tests dans un projet et les lance automatiquement au moindre changement (vous avez une barre en bas à gauche). Si un cas de test est FAIL, vous pouvez également obtenir la cause dans le code en cliquant sur l'erreur.

3. Ne vous Mocker pas ! (Mockito)

Il est fort probable que vous ayez effectué les convertions euro-dollard dans le code avec l'unité de convertion (1 EUR = 1.29 USD). Pour faire mieux, il plus interessant d'utiliser une Classe composée.

Soit donc la Classe *Convertion* qui prends une méthode *Float unit_Convertion (String s)* avec *s* une chaine de convertion "EUR-USD", "USD_EUR" dans notre cas.

Nous ne souhaitons pas coder cette Classe de suite. Il est important d'isoler la classe Money pour ne tester que cette

classe et pas une classe Convertion "codée à l'arrache". Nous allons donc mettre en place un stub (simulation) de cette classe Convertion grâce à Mockito

A

Doc Mockito ICI

Dans Eclipse, pour faire un projet Mockito, le plus simple est d'utiliser Gradle ou Maven.

Dans Eclipse, ajouter le plugin "Gradle Integration" via le marketplace.

Créez un projet Gradle puis modifiez le fichier "build.gradle en ajoutant dans la partie dependency:

```
// Use JUnit test framework
    testImplementation 'junit:junit:4.12'
// ... moreentries

    testCompile 'junit:junit:4.12'
// required if you want to use Mockito for unit tests
    testCompile 'org.mockito:mockito-core:2.7.22'
```

gradle

Faites l'import: import static org.mockito.Mockito.*;

Ajoutez la déclaration:

@Mock Convertion conv; et *@InjectMocks private Money money;* pour indiquer que le mock est injecté dans la Classe Money.

Créez une classe Convertion vide:
public class Convertion {
public double unit_Convertion(String s) {return 0.0;}
}

Ajoutez l'attribut Convertion dans la classe Money et modifiez son code pour utiliser cet attribut. Pour l'instant le Mock se comporte en renvoyant null. bof bof. En ce moment même, vous faites de la non-régression ! Vous testez le fonctionnement du code que vous avez fait via vos tests.

B

Ajoutons du comportement. On va indiquer à Mockito quelle valeur on souhaite retourner lorsque la méthode unit_Convertion est invoquée.

un exemple: when(conv.unit_Convertion("EUR-USD").thenReturn("1.29");

Faites de même dans tous les cas du sujet. Où placer ce code ?

C

Ajoutez dans le Mock que si vous recevez un argument commençant par " " dans la méthode *unit_Convertion* alors vous

renvoyez l'Exception *IllegalArgumentException*

D

Ajoutez la méthode *sub* en commençant par créer les tests.

Stockez la classe *Money*

4. Robustesse

Il semble logique que les montants ne doivent pas être négatifs. Cela fait parti des tests de robustesse. De même, on ne veut pas de devise nulle ou farfelue.

A

Créez deux nouvelles classes de cas de test afin de tester ces points. Combien faut-il de cas de test par classe ?

B

Modifiez le code de la classe *Money* pour que vos tests passent.

5. Couverture de code

A

Donnez la couverture de code obtenue avec Emma dans les questions 2 et 3. Y a t-il une différence?

B

Complétez le code de Money et /ou les cas de test (en précisant les modifications) pour obtenir une couverture de 100%