Cours Génie Logiciel
Steve Hostettler
Assistant: Sunny Avry (sunny.avry@unige.ch)

UNIVERSITÉ DE GENÈVE

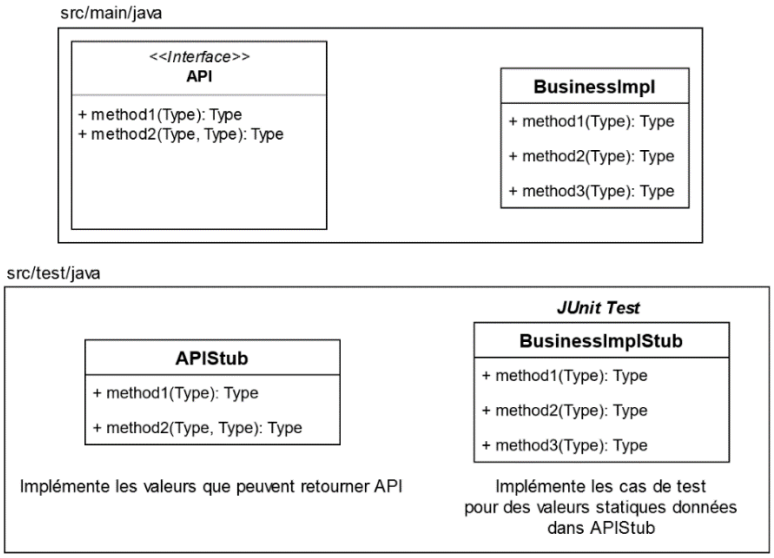
TP07

Software quality and testing 2

A rendre pour la semaine suivant la séance d'exercices.

1 Réaliser un stub avec JUnit

Réaliser dans Eclipse un test impliquant un stub simulant l'utilisation d'une API tel que représenté sur le schéma ci-dessous :



Mise en place du SUT (Software Under Test)

1) Créer un nouveau projet MAVEN
2) Ouvrir le fichier *pom.xml* et copier-coller le code ci-après après </version> pour permettre l'utilisation de JUnit et de Mockito

```
<dependencies>
        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.mockito</groupId>
            <artifactId>mockito-all</artifactId>
            <version>1.10.19</version>
            <scope>test</scope>
        </dependency>
      </dependencies>
```

3) Créer une interface nommée **API**. Cette interface possède la signature de la fonction *public List<String> retrieveCities(String country)* renvoyant la liste des villes d'un pays donné.
4) Créer une classe nommée **BusinessImpl**. Créer une instance *api* de **API** dans cette classe et le constructeur suivant :

```
private API api;

public BusinessImpl(API api) {
	this.api = api;
}
```

5) Créer la fonction *public List<String> retrieveCitiesStartingWithC(String country)*. Dans cette fonction, créer une variable de type *List<String>* nommé *cities* qui utilise la fonction *retrieveCities* de *api* pour récupérer les différentes villes d'un pays donné. Créer ensuite un *ArrayList* nommé *citiesOfInterest* dans lequel seront stockées toutes les villes de *cities* commençant par la lettre c.

Création d'un double de test

On souhaiterait tester la fonction *retrieveCitiesStartingWithC* (qui dépend de la fonction *retrieveCities* de api) en isolation.

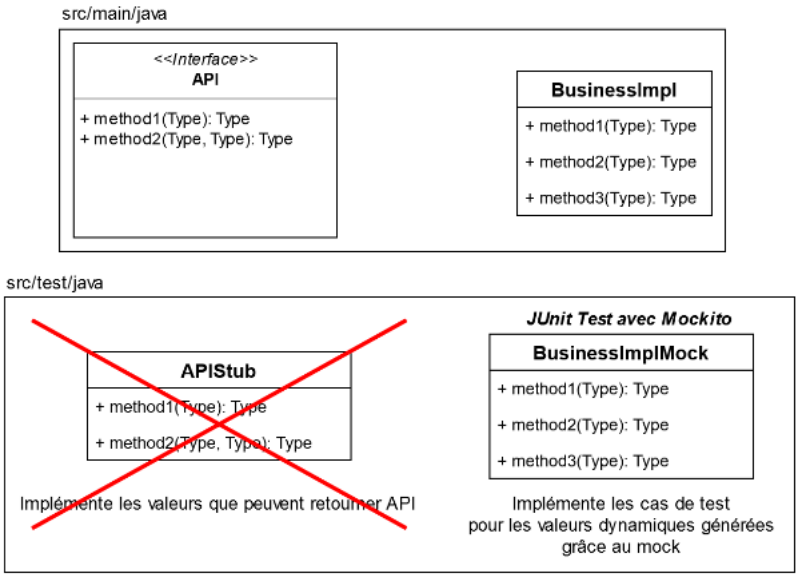
1) Créer une classe nommée **APIStub**. Cette classe implémente l'interface **API**.
2) Recréer dans cette classe une *dummy implementation* de la fonction *public List<String> retrieveCities(String country)* en lui donnant comme valeur de retour une liste de villes prédéfinies, certaines commençant par la lettre c, d'autres non, simulant ce que retournerait l'appel de la fonction réelle.

Création d'un cas de test

1) Créer un cas de test JUnit nommé **BusinessImplStub**.
2) Après @test, créer la méthode *public void testRetrieveCitiesStartingWithC_stub()*.
3) Dans cette méthode, créer une instance de **APIStub** nommée *apiStub* et une instance de **BusinessImpl** nommée *businessImpl* et prenant comme paramètre *apiStub*.
4) Créer une variable de type *List<String>* nommée *citiesOfInterest*. Cette variable stocke les villes commençant par la lettre c retrouvées suite à l'appel de *retrieveCitiesStartingWithC* de *businessImpl***.**
5) Utiliser la fonction *assertEquals(integer a, integer b)* permettant de vérifier que le nombre de villes commençant par la lettre c stockée dans *citiesOfInterest* est identique au nombre de ville commençant par la lettre c de la *dummy implementation* de la fonction *retrieveCities(String country)* de *apiStub*.
6) Lancer le cas de test. S'il est réussi, la fonction *retrieveCitiesStartingWithC* remplit correctement son rôle.

2 Réaliser un mock avec JUnit et Mockito

Réaliser dans Eclipse un test impliquant un mock simulant l'utilisation d'une API tel que représenté sur le schéma ci-dessous :



1) Dans Eclipse, aller dans Window > Preferences > Java > Editor > Content Assist > Favorites et ajouter les imports statiques suivants :

```
org.junit.Assert
org.mockito.BDDMockito
org.mockito.Mockito
org.mockito.Matchers
org.hamcrest.CoreMatchers
```

2) Créer une copie de la classe **BusinessImplStub** et la renommer **BusinessImplMock**.
3) Après @test, renommer la méthode *public void testRetrieveCitiesStartingWithC_stub()* en *public void testRetrieveCitiesStartingWithC_mock()*
4) Créer le mock *apiMock* permettant de simuler le comportement de **API** grâce à la fonction mock() :

```
API apiMock = mock(API.class);
```

apiMock remplace *apiStub*. Conserver *businessImpl* et lui donner comme paramètre le mock *apiMock*.

5) Créer une variable de type *List<String>* nommé *cities* qui simulent les valeurs retournées par *apiMock* (i.e., les différentes villes d'un pays donné)
6) Appeler la fonction when de mockito pour retourner *cities* lorsque l'appel de la fonction *retrieveCities* est réalisé.

```
when(<nomdumock>.function(some parameters)).thenReturn(values);
```

7) Lancer le cas de test. S'il est réussi, la fonction *retrieveCitiesStartingWithC* remplit correctement son rôle.
8) Créer une nouvelle fonction dans **BusinessImpl** permettant de renvoyer le nombre de villes d'un pays donné : *public Integer getNumberOfCities(String country)*.
9) Créer un nouveau cas de test permettant de tester cette fonction avec n'importe quel pays fourni comme paramètre.

3 Travail maison

Choisir un projet Java que vous avez réalisé et répondre aux questions suivantes :

1) Expliquer comment améliorer la couverture de test avec les différents Matchers de Mockito : https://static.javadoc.io/org.mockito/mockito-core/1.10.19/org/mockito/Matchers.html
2) Tester quatre fonctions de votre projet, deux avec un stub, deux autres avec un mock.
3) Quels sont les avantages de l'utilisation de JUnit pour tester ses fonctions ? Quels sont les conditions où l'utilisation d'un stub est appropriée ? Quand est-ce que l'utilisation d'un mock est plus intéressante ? Penser notamment à la maintenance lorsque de nouvelles fonctions sont ajoutées à la classe ou l'interface dont vous simulez les données de test et à la dynamicité de la génération de ces données.

À rendre

Un rapport pdf contenant les éléments suivants :

- Les réponses aux questions 1 et 3 de la partie Travail maison
- Un copié-collé de votre classe de test JUnit implémentant vos quatre tests (deux stubs, deux mocks). Indiquer en commentaires le but de vos différents tests.