IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

TP - EASYMOCK

OBJECTIFS

L'objectif de cette séance est de découvrir l'intérêt des simulacres (« mocks ») pour le test de logiciel, et d'expérimenter `EasyMock`, un outil facilitant leur mise en œuvre en Java.

TEST DE LA CLASSE `MonMenu`

Importez dans Eclipse le projet de l'archive `Menu.tar.gz` disponible sur Moodle.

Ce projet comporte la classe `MonMenu` dont la méthode `menu()` a la particularité d'appeler la méthode `un_entier()` prévue pour le type (cf interface) `IGenerateur`. Cette méthode permet typiquement à l'utilisateur de préciser son choix de manière interactive (IHM) parmi plusieurs propositions ; elle renvoie un entier correspondant au choix de l'utilisateur. Vu qu'elle nécessite de l'interaction avec l'utilisateur, le test de `MonMenu` ne pourra *pas* être automatisé ! Dans le cas où la classe implémentant l'interface `IGenerateur` ne serait pas encore écrite, le test de `MonMenu` serait alors également impossible.

Nous allons contourner ces difficultés en utilisant des « simulacres » (« mock » en anglais). Il s'agit de fabriquer - à moindres frais - un objet ayant le même (genre de) comportement que celui de l'objet qu'on ne peut/veut pas utiliser, afin de pouvoir exécuter du code qui en dépend, même en l'absence de celui-ci. On peut ainsi tester au plus tôt du code même s'il appelle des composants non disponibles ou qu'on ne souhaite pas mobiliser. Bien évidemment, pour être produit à moindres frais, ce simulacre n'implémentera pas vraiment le traitement, mais ne servira qu'à faire semblant. Dans notre exemple, on pourra écrire un simulacre de la méthode `un_entier()` qui renvoie directement un entier approprié pour nos tests (un entier entre 1 et 3), mais sans consulter pour cela l'utilisateur, permettant ainsi l'automatisation de nos tests.

Créez une classe `SimulacreGenerateur` implémentant `IGenerateur` et ayant une méthode `un_entier()` qui renvoie directement la valeur `2`.

Ajoutez un `main()` à la classe `MonMenu` utilisant ce simulacre pour exécuter la méthode `menu()`. Exécutez ce `main()`.

Bien évidemment, comme le simulacre renvoie systématiquement la même valeur 2, la méthode `menu()` boucle :

Arrêtez l'exécution (cf bouton rouge « `Terminer` »)

Vous pouvez retenter l'expérience en demandant à Emma de vous montrer les parties de code exécutées :

Vérifiez avec Emma que seule la partie de `menu()` correspondant à la valeur `2` a été exécutée.

Notre simulacre est trop « basique » pour nous permettre d'aller plus loin dans le test.

UTILISATION DE EASYMOCK

Des outils ont été élaborés pour faciliter l'élaboration de simulacres qui ont un comportement « programmé ». Nous allons utiliser un de ces outils : `EasyMock` (cf http://easymock.org). Cela permettra de fabriquer un objet simulacre directement, sans même écrire de classe spécialement pour cela. Voyons le principe d'utilisation de `EasyMock`.

Principe d'utilisation de EasyMock

« Installation »

Pour pouvoir utiliser `EasyMock` dans le projet, ajouter `easymock.jar` (disponible sur Moodle) au `classpath` de votre projet (`clic droit → Properties → Java build path → classpath → add external JARs`).

Déclaration et instanciation d'un nouveau mock

Pour pouvoir utiliser le simulacre, il faut déjà lui donner une existence : on crée directement un objet, sans créer de classe pour cela :

```
MaClasseASimuler mock;
mock = EasyMock.createMock(MaClasseASimuler.class);
```

Définition du comportement attendu

Puis on définit le comportement de cet objet, un peu à la manière d'un apprentissage supervisé : pour chaque méthode qui sera appelée à l'exécution, on indique les valeurs attendues en paramètres d'appel (s'il y a des paramètres) et la valeur que devra retourner l'objet simulacre.

```
EasyMock.expect(mock.laMethodeAppelee(lesArgumentsAttendus)).andReturn(valeurARetourner);
```

Dans le cas où aucune valeur n'est renvoyée, on indique simplement l'appel attendu, suivi d'une instruction indiquant que cet appel doit se faire :

```
mock.laMethodeAppelee(lesArgumentsAttendus);
EasyMock.expectLastCall();
```

On peut également demander au simulacre une levée d'exception :

```
EasyMock.expect(mock.laMethodeAppelee(lesArguments)).andThrow(uneException[1]);
```

Il existe plusieurs autres façons[2] d'entraîner le simulacre : se reporter à la documentation `EasyMock` si besoin. Quand on a listé tous les appels prévus, on termine l'apprentissage pour pouvoir utiliser le simulacre :

```
EasyMock.replay(mock);
```

Utilisation

On peut ensuite écrire le code qui appelle ce simulacre. Si les appels effectifs se font bien dans l'ordre appris, avec les paramètres attendus, le simulacre renverra les valeurs prévues. Sinon, une erreur sera signalée par `EasyMock`.

Finalisation

On peut éventuellement vérifier explicitement que tous les appels prévus ont bien été effectués :

```
EasyMock.verify(mock) ;
```

Application au test de MonMenu

Vous allez utiliser `EasyMock` pour créer un simulacre de `un_entier()` qui renverra successivement différentes valeurs entières pertinentes que vous voulez tester, et terminera par la valeur `0` pour terminer `mon_menu()`. Votre code pourra bien évidemment s'appuyer également sur `JUnit`.

Écrivez un code de test pour la classe `MonMenu` qui mobilise un simulacre `EasyMock` de `un_entier()` afin d'obtenir 100 % de couverture de branches pour `menu()`.

S'il vous reste du temps, vous pourrez viser des couvertures de test plus ambitieuses (cf couverture de chemins).

1 `UneException` est typiquement fournie sous la forme : `new LExceptionDesiree()`
2 À noter que tout ceci évolue avec les versions de `EasyMock`.

TEST DE LA CLASSE `MesFonctions`

La méthode `menu()` invoque les fonctions définies dans la classe `MesFonctions.`

Créez une classe MesFonctionsTest qui procure 100 % de *couverture de test d'instructions* pour ces trois fonctions.

Rappel : pour connaître la couverture de test, exécuter le programme de test via Emma, puis visualiser les taux de couverture : soit via la coloration de code source, soit en demandant la couverture dans les propriétés de la classe testée.