Tests orientés objets
Mocking - Mockito

Philippe Collet

L3 MIAGE

2016-2017

Plan

- Mocking
- Mockito

Définition

- Mock = Objet factice
- les mocks (ou Mock object) sont des objets simulés qui reproduisent le comportement d'objets réels de manière contrôlée
- On teste ainsi le comportement d'autres objets, réels, mais liés à un objet inaccessible ou non implémenté
- Cet objet est remplacé par un mock

Définition(s)

- dummy (pantin, factice) : objets vides qui n'ont pas de fonctionnalités implémentées
- stub (bouchon) : classes qui renvoient en dur une valeur pour une méthode invoquée
- fake (substitut, simulateur) : implémentation partielle qui renvoit toujours les mêmes réponses selon les paramètres fournis
- spy (espion) : classe qui vérifie l'utilisation qui en est faite après l'exécution
- mock (factice) : classes qui agissent comme un stub et un spy

Exemple d'utilisation

- Comportement non déterministe (l'heure, un senseur)
- Initialisation longue (BD)
- Classe pas encore implémentée ou implémentation changeante, en cours
- Etats complexes difficiles à reproduire dans les tests (erreur réseau, exception sur fichiers)
- Pour tester, il faudrait ajouter des attribut ou des méthodes

Principe

- Un mock a la même interface que l'objet qu'il simule
- L'objet client ignore s'il interagit avec un objet réel ou un objet simulé
- La plupart des frameworks de mock permettent
  - De spécifier quelles méthodes vont être appelées, avec quels paramètres et dans quel ordre
  - De spécifier les les valeurs retournées par le mock

Université Nice Sophia Antipolis

Mockito

http://site.mockito.org/

Eléments de cours de M. Nebut lifl.fr

Mockito

- Générateur automatique de doublures
- Léger
  - Focalisation sur le comportement recherché et la vérification après l'exécution
- Simple
  - Un seul type de mock
  - Une seule manière de les créer

Principes

- Fonctionnement en mode espion (spy) :
  - Création des mocks
    - méthode mock ou annotation @mock
  - Description de leur comportement
    - Méthode when
  - Mémorisation à l'exécution des interactions
    - Utilisation du mock dans un code qui teste un comportement spécifique
  - Interrogation, à la fin du test, des mocks pour savoir comme ils ont été utilisés
    - Méthode verify

import static org.mockito.Mockito.*

Création

- Par une interface ou une classe (utilisation de .class)
  - UneInterface mockSansNom = mock(UneInterface.class);
  - UneInterface mockAvecNom =
    mock(UneInterface.class, "ceMock");
  - @Mock UneInterface ceMock;
- Comportements par défaut
  - assertEquals("ceMock", monMock.toString());
  - assertEquals("type numerique : 0 ", 0,
    monMock.retourneUnEntier());
  - assertEquals("type booleéen : false",
    false, monMock.retourneUnBooleen());
  - assertEquals("type collection : vide",
    0, monMock.retourneUneList().size());

Stubbing

- Pour remplacer le comportement par défaut des méthodes
- Deux possibilités
  - Méthode qui a un type de retour :
    - when + thenReturn ;
    - when + thenThrow ;
  - Méthode de type void :
    - doThrow + when ;

Stubbing
retour d'une valeur unique

```
// stubbing
when(monMock.retourneUnEntier()).thenReturn(3);

// description avec JUnit
assertEquals("une premiere fois 3", 3, monMock.retourneUnEntier());
assertEquals("une deuxieme fois 3", 3, monMock.retourneUnEntier());
```

Stubbing
valeurs de retour consécutives

```
// stubbing
when(monMock.retourneUnEntier()).thenReturn(3, 4, 5);

// description avec JUnit
assertEquals("une premiere fois : 3", 3, monMock.retourneUnEntier());
assertEquals("une deuxieme fois : 4", 4, monMock.retourneUnEntier());
assertEquals("une troisieme fois : 5", 5, monMock.retourneUnEntier());

when(monMock.retourneUnEntier()).thenReturn(3, 4);
// raccourci pour      .thenReturn(3).thenReturn(4);
```

Stubbing
Levée d'exceptions

```
public int retourneUnEntierOuLeveUneExc() throws BidonException;
// stubbing
when(monMock.retourneUnEntierOuLeveUneExc()).thenReturn(3)
                    .thenThrow(new BidonException());

// description avec JUnit
assertEquals("1er appel : retour 3",
                3, monMock.retourneUnEntierOuLeveUneExc());

try {
    monMock.retourneUnEntierOuLeveUneExc(); fail();
} catch (BidonException e) {
    assertTrue("2nd appel : exception", true);
}
```

Levée d'exception + méthode void = doThrow

Remarques

- Les méthodes equals() et hashcode() ne peuvent pas être *stubbées*
- Un comportement de mock non exécuté ne provoque pas d'erreur
- Il faut utiliser *verify*
  - Quelles méthodes ont été appelées sur un
  - Combiendefois, avec quels paramètres, dans quel ordre
- Une exception est levée si la vérification échoue, le test échouera aussi

Verify

- Méthode appelée une seule fois :
  - verify(monMock).retourneUnBooleen();
  - verify(monMock, times(1)).retourneUnBooleen();
- Méthode appelée au moins/au plus une fois:
  - verify(monMock, atLeastOnce()).retourneUnBooleen();
  - verify(monMock, atMost(1)).retourneUnBooleen();
- Méthode jamais appelée :
  - verify(monMock, never()).retourneUnBooleen();
- Avec des paramètres spécifiques :
  - verify(monMock).retourneUnEntierBis(4, 2);

Verify

- import org.mockito.InOrder;
- Pour vérifier que l'appel (4,2) est effectué avant l'appel (5,3) :
  - InOrder ordre = inOrder(monMock);
  - ordre.verify(monMock).retourneUnEntierBis(4, 2);
  - ordre.verify(monMock).retourneUnEntierBis(5, 3);
- Avec plusieurs mocks :
  - InOrder ordre = inOrder(mock1, mock2);
  - ordre.verify(mock1).foo();
  - ordre.verify(mock2).bar();

Espionner un objet classique

- Pour espionner autre chose qu'un objet mock (un objet « réel »):
  - Obtenir un objet espionné par la méthode spy :

    ```
    List list = new LinkedList();
    List spy = spy(list);
    ```

  - Appeler des méthodes « normales » sur le spy :

    ```
    spy.add("one");
    spy.add("two");
    ```

  - Vérifier les appels à la fin :

    ```
    verify(spy).add("one");
    verify(spy).add("two");
    ```

Matchers

- Mockito vérifie les valeurs en arguments en utilisant equals()
- Assez souvent, on veut spécifier un appel dans un « when » ou un « verify » sans que les valeurs des paramètres aient vraiment d'importance

- On utilise pour cela des *Matchers (import org.mockito.Matchers)*

  - when(mockedList.get(**anyInt**())).thenReturn("element");

  - verify(mockedList).get(**anyInt**());

Matchers

- Les Matchers très souvent utilisés :
  - any() , static <T> T anyObject()
  - anyBoolean(), anyDouble(), anyFloat() , anyInt(), anyString()…
  - anyList(), anyMap(), anyCollection(), anyCollectionOf(java.lang.Class<T> clazz) ,anySet(), <T> java.util.Set<T> anySetOf(java.lang.Class<T> clazz)
  - …

- Attention ! Si on utilise des matchers, tous les arguments doivent être des matchers :
  - verify(mock).someMethod(anyInt(), anyString(), "third argument");
  - n'est pas correct !