Tests structurels en Java

Fabrice AMBERT, Fabrice BOUQUET, Fabien PEUREUX,
Jean-Marie GAUTHIER, Alexandre VERNOTTE
prenom.nom@femto-st.fr

ENVOL 2014
La Londe-les-Maures
18-21 novembre 2014

Outils mis en œuvre

Anatomie d'un test unitaire

```
@Test       Annotation désignant la méthode comme un test
public void testXXX() {
        //Define
                    Instructions de mise en contexte
        //When
                     Instruction sous test
        //Then
                     Observation et vérification de l'oracle
}
```

Tester l'apparition d'une exception

Forme simple

```
@Test (expected = ClasseException.class)
public void testXXX() {
	//Define

	//When

}
```

Tester l'apparition d'une exception

Forme avancée

```
@Rule
Public ExpectedException thrown = ExpectedException.none();
@Test
public void testXXX() {
        // ici l'exception fait échouer le test
    thrown.expect(ExceptionAttendue.class);
        // ici l'exception est attendue
}
```

Quelques annotations

@Ignore

```
@Ignore
@Test(expected = ClasseException.class)
public void testXXX() {
    //Define

    //When

}
```

Lors de l'exécution de la suite, le test est ignoré mais est mentionné dans le rapport d'exécution

Quelques annotations - 2

@Before @After

```
@Before
public void setUp() {

}

@After
public void tearDown() {

}
```

Cette méthode est exécuté avant chaque test de la suite

Celle ci après chaque test de la suite

Test en isolation

```
public class Location {
private Film film ;
private Client client ;
...
public float montant(int duree) {
      if (client.getCat() == PRIVILEGE)
            return film.prixJour()*(duree-1) ;
      else ...
```

```
public class Film {
private Categorie categorie ;
private String titre ;
...
public float prixJour() {
      switch (categorie) {
        case Categorie.NOUVEAUTE :
        return categorie.prixBase() * ... ; ...
```

Test en isolation

```
@Test
public void testMontant() {
        Film film = Mockito.mock(Film.class) ;
        Mockito.when(film.prixJour()).thenReturn(3.5) ;
        Client client = Mockito.mock(Client.class) ;
        Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;
        Location loc = new Location(film, client) ;

        Assert.assertEquals(3.5, loc.montant(2)) ;
}
```

Le mock retourne les réponses attendues par la classe sous test sans faire appel à la classe mockée

Test en isolation

```
@Test
public void testMontant() {
        Film film = Mockito.mock(Film.class) ;
        Mockito.when(film.prixJour()).thenReturn(3.5) ;
        Client client = Mockito.mock(Client.class) ;
        Mockito.when(client.getCat()).thenReturn(PRIVILEGE) ;
        Location loc = new Location(film, client) ;

        loc.montant(2)) ;

        Mockito.verify(film).prixJour();
}
```

Le mock mémorise les appels qui lui sont fait. On peut ensuite l'interroger sur les invocations auxquelles il a répondu.

Merci pour votre attention…