Ateliers java - séance 7

Objectifs :

→ Comprendre comment tester les associations entre classes.
→ Être capable d'écrire des tests unitaires en écrivant des stubs et des mock objects pour les classes dont dépend l'unité testée.

Thèmes abordés :

- Mock et stub objects

Votre travail aujourd'hui consiste à tester la classe `Division` de l'énoncé de la séance 6. Pour des raisons de temps, nous testons uniquement l'association entre la classe `Equipe` et la classe `Division` dans le sens `Division` vers `Equipe`.

Récupérez le projet AJ_2017_seance7. Il contient les interfaces du domaine, la classe `DivisionImpl` ainsi que la classe `ClubStub`.

Stubs

Un plan de tests a été mis au point. Dans celui-ci l'état représente le nombre d'équipes de la division.

TC#	État	Méthode	Paramètres	Résultat attendu	État	Valeur retour	Exception
1	0	ajouterEquipe	Equipe valide	L'équipe est ajoutée	1	true	
2	1	ajouterEquipe	Equipe valide	L'équipe est ajoutée	2	true	
3	2	ajouterEquipe	Equipe déjà présente	L'équipe n'est pas ajoutée	2	false	
4	11	ajouterEquipe	Equipe valide	L'équipe est ajoutée	12	true	
5	12	ajouterEquipe	Equipe valide	L'équipe n'est pas ajoutée	12	false	
6	0	ajouterEquipe	Equipe non valide (autre division)	Aucun changement d'équipe	0	false	
7	0	ajouterEquipe	Equipe valide (déjà la division)	L'équipe est ajoutée	1	true	
8	2	supprimerEquipe	Equipe non présente	L'équipe n'est pas supprimée	2	false	
9	2	supprimerEquipe	Equipe déjà présente	L'équipe est supprimée	1	true	
10	1	supprimerEquipe	Equipe déjà présente	L'équipe est supprimée	0	true	
11	0	supprimerEquipe	Equipe non présente	L'équipe n'est pas supprimée	0	false	
12	0	contientEquipe	Equipe non présente		0	false	

13	1	contientEquipe	Equipe non présente		1	false	
14	2	contientEquipe	Equipe déjà présente		2	true	
15	0	ajouterEquipe	null	L'équipe n'est pas ajoutée	0		IllegalArgumentException
16	1	supprimerEquipe	null	L'équipe n'est pas supprimée	1		IllegalArgumentException
17	1	contientEquipe	null		1		IllegalArgumentException
18	0	equipes			0		MinimumMultiplicityException
19	1	equipes			1		MinimumMultiplicityException
20	11	equipes			11		MinimumMultiplicityException
21	12	equipes			12	liste des equipes	

Afin d'implémenter ces vrais tests unitaires, nous vous demandons d'écrire le stub pour la classe `Equipe;` le stub pour la classe `Club` étant déjà donné. La classe `Division` nécessite des objets des classes `Equipe` et la classe `Equipe` nécessite des objets de la classe `Club`.

Avant d'écrire le stub il faut se demander ce qu'il doit retourner pour chaque méthode.

Dans Eclipse, générez le stub `EquipeStub` à partir de l'interface `Equipe`. Vous devez aussi configurer les valeurs de retours des méthodes utilisées par `Division` dans le constructeur du stub afin de pouvoir diriger votre test. Le constructeur prend donc en paramètre : le `numero`, la `division` et le `club` ainsi que les valeurs booléennes de retour des méthodes `supprimerDivision`, `maximumDivisionAtteint` et `enregistrerDivision`.

Écrivez ensuite les jeux de tests dans une classe `TestDisivionStub`. Nommez vos méthodes de test en fonction des cas décrits dans le plan de tests. Par exemple: `TestDivisionTC1`.

Vous pouvez écrire une méthode `private` s'appelant `amenerALEtat(int etat, Division division)` afin d'amener un objet `division` à l'état à tester (ce qui veut dire au nombre d'équipes désiré pour le test).

N'oubliez pas que vous devez vérifier que le nouvel état est atteint et que le résultat est bien celui attendu.

Mocks objects

Les tests que nous allons effectuer maintenant portent sur les transitions d'états ; il faut tester si les méthodes interagissent correctement avec les autres objets et leur donnent les valeurs attendues. Implémentez les tests du comportement des méthodes avec des mocks objects.

Nous avons déjà établi un plan de tests pour la méthode `ajouterEquipe(Equipe equipe)` :

TC#	Etat	Méthode	Paramètres	Résultat attendu	Etat	Valeur retour	Objet participant	Méthode appelée	Paramètres
1	0	ajouterEquipe	Equipe valide	L'équipe est ajoutée	1	true	equipe	enregistrer Division	la division
2	1	ajouterEquipe	Equipe valide	L'équipe est ajoutée	2	true	equipe	enregistrer Division	la division
3	12	ajouterEquipe	Equipe valide	L'équipe n'est pas ajoutée	12	false	equipe	aucune	
4	0	ajouterEquipe	Equipe non valide (autre division)	Aucun changement d'équipe	0	false	equipe	aucune	
5	0	ajouterEquipe	Equipe valide (division correcte déjà définie)	L'équipe est ajoutée	1	true	equipe	enregistrer Division	la division

Comme nous devons vérifier que la méthode `ajouterEquipe` de la classe `Division` met bien en place les deux côtés de l'association en appelant la méthode `enregistrerDivision` de l'objet `equipe`, nous allons devoir construire un (ou plusieurs) mock(s) object(s) pour la classe `Equipe`.

L'objectif d'un mock object est de vérifier que l'on appelle bien la ou les méthodes attendues avec le ou les bons paramètres.

Comme il n'y a pas de comportement attendu à vérifier sur les objets de `Club`, vous pouvez continuer à utiliser des stubs pour cette classe.

Pour implémenter la classe `EquipeMock`, il faut

1) copier-coller le `EquipeStub`
2) conserver dans l'objet :
   - Le(s) paramètre(s) attendu(s)
     - `enregistrerDivisionExpectedParam`
   - Les appels de méthodes attendus
     - `boolean enregistrerDivisionCallExpected`
   - Les appels de méthodes effectivement effectués
     - `boolean enregistrerDivisionCall`
3) écrire une méthode qui va vérifier que les méthodes effectivement appelées sont celles attendues (conservées dans l'objet)
   - Appelez cette méthode `verify`
   - Invoquez cette méthode après chaque test.
4) écrire les jeux de tests dans la classe `TestDivisionMock`.

Vous devez ensuite établir un plan de tests pour la méthode `supprimerEquipe(Equipe equipe)` de la classe `Division` et compléter la classe `TestDivisionMock` afin de réaliser les tests correspondant à ce plan de tests.