TP2 : Test en isolation avec Mockito

S2 Master1 informatique
TP SVL – 2017-2018

Compétences : tester en isolation en utilisant une bibliothèque de mocks automatisée (Mockito pour java). Faire la différence entre test d'état et test des interactions. Donner des noms clairs aux tests. Choisir les tests en suivant une approche fonctionnelle.

Rendre une archive contenant vos sources et vos tests, avec votre nom dans chaque fichier, et un README contenant aussi vos noms.

Exercice 1 : Jeu de dés

On s'intéresse à des jeux de hasard et d'argent du type casino en ligne. Le sujet n'est pas très réaliste.

1.1 : Spécification informelle

Le jeu qui nous intéresse, le « jeu du 7 » se joue avec 2 dés et une banque qui gère les pertes et les gains. Le joueur qui veut jouer propose une mise, qui est la valeur de son pari. Les dés sont du modèle classique à tirage aléatoire entre 1 et 6. La banque encaisse les paris et décaisse les gains. En cas de gains trop importants, la banque peut n'être plus solvable, alors le jeu ferme.

La règle du jeu est la suivante. On ne peut jouer qu'à un jeu ouvert. Le joueur propose interactivement une mise, puis est débité du montant de son pari. S'il est insolvable, le jeu s'arrête là. Sinon la mise est encaissée par la banque. Ensuite les 2 dés sont lancés. Si la somme des lancers ne vaut pas 7, le joueur a perdu sa mise et le jeu s'arrête là. Si la somme des lancers vaut 7, alors le joueur gagne : la banque paye deux fois la mise, somme créditée au joueur. Le jeu consulte la banque pour savoir si elle est encore solvable, et si ce n'est pas le cas il ferme.

1.2 : Sujet

L'interface de la classe `Jeu` est la suivante :

```
public Jeu(Banque labanque);
public void jouer(Joueur joueur, De de1, De de2) throwsJeuFermeException;
public void fermer();
public boolean estOuvert();
```

Le sujet consiste à tester **en isolation** la méthode `jouer`. Le test sera donc purement unitaire au niveau de la classe. **On ne demande pas d'écrire ni de tester les autres classes** sauf dernière question : limitez-vous à des **interfaces** et utilisez des **doublures**.

On propose les interfaces suivantes :

```
public interface De {   public int lancer(); }
public interface Joueur {
    public int mise(); // on suppose que mise positive
    public void debiter(int somme) throws DebitImpossibleException;
    public void crediter(int somme);
}
public interface Banque {
    public void crediter(int somme);
    public boolean est_solvable();
    public void debiter(int somme);
}
```

Q 1.1 : Quels objets dont dépend la classe `Jeu` sont forcément mockés dans un test de automatiser de `jouer` ? Pourquoi ? Répondre dans un `README`. □

Q 1.2 : Lister les scénarios JUnit que vous allez écrire pour tester `jouer`, en les décrivant en français dans le `README`. Écrire les cas nominaux et les cas exceptionnels. □

Q 1.3 : Écrire le code Java pour Jeu. □

Q 1.4 : Commencer par écrire le test le plus simple : le cas où le jeu est fermé. Est-ce un test d'état ou un test des interactions ? Répondre dans le `README`. □

Q 1.5 : Tester le cas où le joueur est insolvable. Comment tester que le jeu ne touche pas aux dés ? Est-ce un test d'état ou un test des interactions ? Répondre dans le `README`. □

Q 1.6 : Continuer avec les autres scénarios. □

La banque possède un fond, et un fond minimum à ne pas dépasser (initialisés dans le constructeur). Si elle passe en dessous de ce niveau suite à un pari gagnant, le gain est quand même donné au joueur, mais le jeu ferme car la banque n'est plus solvable (pas très réaliste, mais bon).

Q 1.7 : Écrire une implémentation pour la banque, et écrire à nouveau un des tests impliquant la banque, en l'intégrant cette fois (donner un titre parlant). Méditer sur la différence entre les 2 tests. □