TP DE TEST : JMOCK

AHOUNOU Folabi Thierry

Familiarisation avec le Framework Jmock

Université de Rouen

Master1 GIL

11/04/2013

Table des matières

Introduction 2
Conception 2
Analyse 2
Diagramme de classe 3
Présentation du Framework JMock 4
Conclusion 4
Scenario de tests sur le tp de jmock 4

Introduction

Le but de ce TP est de nous familiariser avec le Framework JMock à travers la programmation d'un jeu de dés impliquant une banque et des joueurs. La méthode utilisée pour ce TP est le TDD.

Pour répondre au problème une phase d'analyse a été faite avant de passer à la conception.

Dans ce rapport on énoncera l'analyse faite et le diagramme de classe qui en découle. Suivront une brève présentation du Framework JMock, une petite conclusion et les scénarios de test.

Conception

Analyse

Pour développer ce TP on n'a pas eu à chercher les classes nécessaires. En effet ils étaient plus ou moins identifiés dans le sujet.

L'analyse du sujet a conduit donc à énumérer quatre entités différentes à savoir :

- Un joueur représenté ici par l'interface IPlayer
- Une banque représentée ici par l'interface IBank
- Une interface IGame et la classe l'implémentant Game
- Une interface IDice pour représenter les dés.

A ces quatre classes on ajouté deux classes d'exception BankRuptException et GameIsCloseException. BankRuptExcetption sert à gérer le cas où la banque saute c'est-à-dire quand le montant minimum de la banque a été violé. Quand la banque saute, elle est donc fermée. La classe GameIsCloseException est donc là pour ce cas.

Le but a été vraiment, de séparer chaque entité afin de pouvoir minimiser le lien entre les différentes classes.

Chaque interface ou classe codé a été clairement commenté afin de permettre une meilleure compréhension du code.

Diagramme de classe

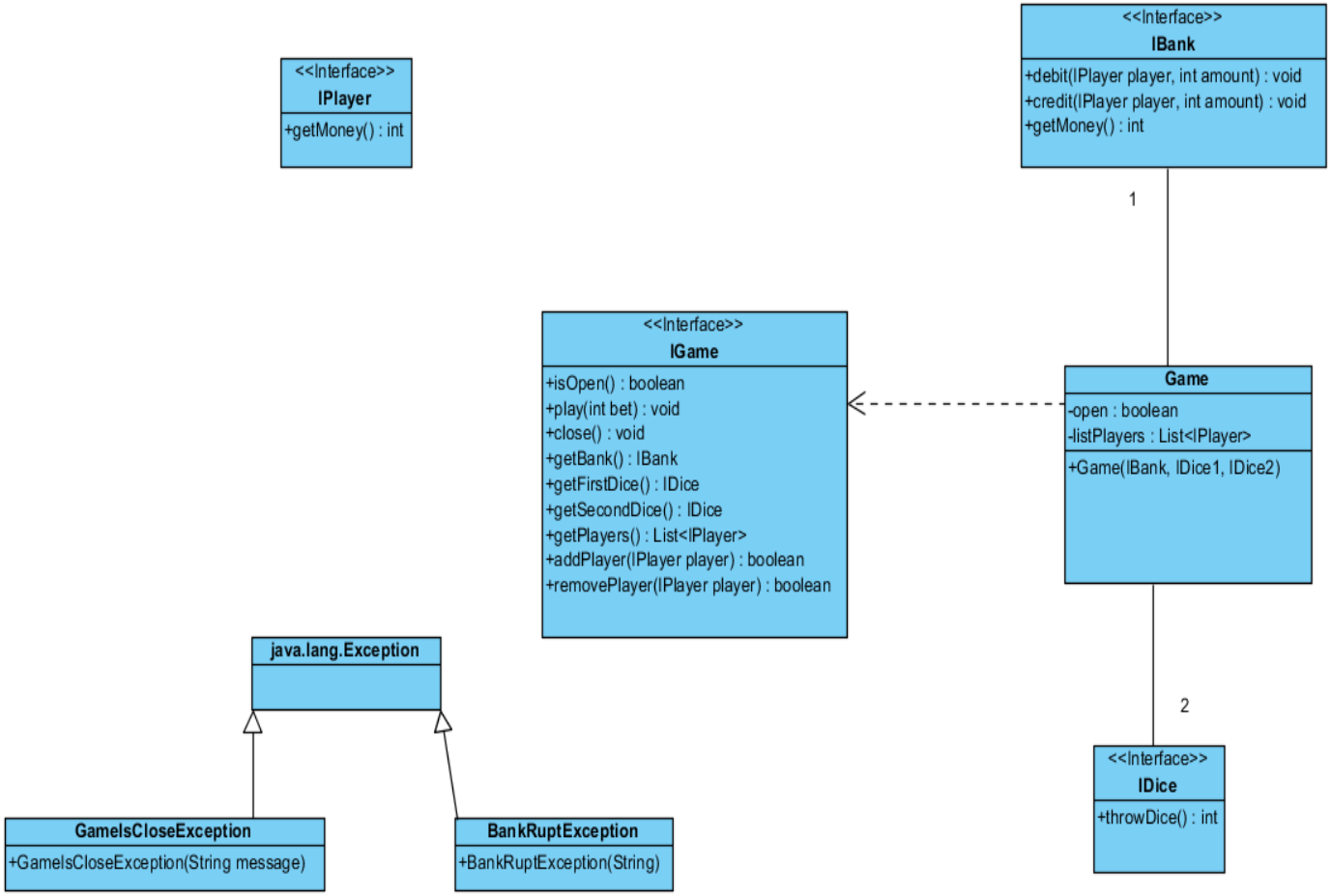

Figure 1 : Diagramme de classe du TP

Présentation du Framework JMock

JMock est un Framework Java qui permet de réaliser des tests unitaires. Elle nécessite d'avoir un code bien structuré. En effet elle repose sur l'utilisation d'interfaces.

JMock permet de faire du TDD. Il permet de simuler des objets qui sont utilisés à la place des vrais objets. Il s'agit d'une approche pratique car on peut travailler sur un projet sans avoir les vraies implémentations des classes nécessaires. JMock permet de définir le comportement attendu. La seule chose à faire est de définir les interfaces qu'il va simuler.

Conclusion

Ce TP nous a permis de découvrir le Framework JMock et de nous habituer à écrire des tests en utilisant la méthode de TDD. Par ailleurs les tests ont permis d'assurer le bon fonctionnement du jeu.

Scenario de tests sur le tp de jmock

Identifiant de projet	TP6 JMOCK		
Nom AAT	Game.play(int)	**Version**	1.0
Phase de test	Unitaire	**Date**	03/04/13

Identifiant de test	testGameIsClose()
Objectif	Levée d'exception car le jeu est fermé
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour bank, dice et player `game = new Game(bank, dice, dice);` `// fermeture du jeu, open == false` `game.close();` `game.play(20);`
Résultat attendu	Lève l'exception GameIsCloseException

Identifiant de test	testBetNegativeInfiniteValue()
Objectif	Jeu de test aux limites de la mise (Integer.MIN_VALUE)
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `game = new Game(bank, dice, dice);` `game.play(Integer.MIN_VALUE);`
Résultat attendu	Lève l'exception IllegalArgumentException

Identifiant de test	testBetNegativeValue()
Objectif	Jeu de test avec une mise negative et la mise comprise entre (Integer.MIN_VALUE et 0)
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `game = new Game(bank, dice, dice);` `game.play(-30);` `// random negative` `int random = (int)(Math.random() * (0-Integer.MIN_VALUE)) + Integer.MIN_VALUE;` `game.play(random);`
Résultat attendu	Lève l'exception IllegalArgumentException

Identifiant de test	testBetBadValue()
Objectif	Jeu de test avec une mise = 0
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `game = new Game(bank, dice, dice);` `game.play(0);`
Résultat attendu	Lève l'exception IllegalArgumentException

Identifiant de test	testBetPositiveInfiniteValue()
Objectif	Jeu de test avec une mise positivement infinie
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `game = new Game(bank, dice, dice);` `game.play(Integer.MAX_VALUE + 1);` `int random = (int)(Math.random() * (Integer.MAX_VALUE-0)) + 0;` `game.play(Integer.MAX_VALUE + random);`
Résultat attendu	Lève l'exception IllegalArgumentException

Identifiant de test	testGoodBet ()
Objectif	Jeu de test avec une bonne mise
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player

Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `game = new Game(bank, dice, dice);` `game.play(20);` `game.play(Integer.MAX_VALUE - 1);`
Résultat attendu	Pas d'exception

Identifiant de test	testPlayerLose ()
Objectif	Jeu de test avec un joueur perdant
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `context.checking(new Expectations(){ {` `  // player a 20€` `  oneOf(player).getMoney(); will(returnValue(20));` `  // lancer des des` `  exactly(2).of(dice).throwDice();` `  will(onConsecutiveCalls(returnValue(4), returnValue(1)));` `  // le joueur a perdu, la banque le debite de 10€` `  oneOf(bank).debit(player, 10);` `  oneOf(bank).getMoney(); will(returnValue(50 + 10));` `} });` `game = new Game(bank, dice, dice);` `game.addPlayer(player);` `game.play(10);` `context.assertIsSatisfied();` `assertTrue(game.isOpen());`
Résultat attendu	Scenario vérifié. Le joueur a 20€ et perd. Il est donc débité du montant de la mise qui est de 10€. Le jeu reste ouvert.

Identifiant de test	testPlayerWin ()
Objectif	Jeu de test avec un joueur gagnant
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `context.checking(new Expectations(){ {` `  // player a 20€` `  oneOf(player).getMoney(); will(returnValue(20));` `  // lancer des des` `  exactly(2).of(dice).throwDice();` `  will(onConsecutiveCalls(returnValue(4), returnValue(3)));` `  // le joueur a gagne, la banque le credite de 2 * 20€` `  oneOf(bank).credit(player, 20*2);` `  oneOf(bank).getMoney(); will(returnValue(980 - (20*2)));` `} });` `game = new Game(bank, dice, dice);` `game.addPlayer(player);` `game.play(20);` `context.assertIsSatisfied();` `assertTrue(game.isOpen());`
Résultat attendu	Scenario vérifié. Le joueur joue et gagne avec au départ 20€ à son actif. Il gagne 2 * le montant de la mise donc 20 * 2. Le jeu reste ouvert.

Identifiant de test	testBankRupt ()
Objectif	Jeu de test avec une banque qui saute.
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `context.checking(new Expectations(){ {` `  // player a 20€` `  oneOf(player).getMoney(); will(returnValue(20));` `  // lancer des des` `  exactly(2).of(dice).throwDice();` `  will(onConsecutiveCalls(returnValue(4), returnValue(3)));` `  // le joueur a gagne, la banque le credite de 2 * 20€` `  oneOf(bank).credit(player, 50*2);` `  oneOf(bank).getMoney(); will(returnValue(50 - (50*2)));` `} });` `game = new Game(bank, dice, dice);` `game.addPlayer(player);` `game.play(50);` `context.assertIsSatisfied();`
Résultat attendu	Scenario vérifié. Le joueur joue et gagne avec au départ 20€ à son actif. Il gagne 2 * le montant de la mise donc 50 * 2. La banque saute du fait qu'elle avait au départ 50€. L'exception BankRuptException est levée.

Identifiant de test	testPlayersWinOneLose ()
Objectif	Jeu de test avec plusieurs joueurs qui gagnent et un perd.
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player, player2, player3
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player `// player 1 joue et gagne` `context.checking(new Expectations(){ {` `  // player1 a 20€` `  oneOf(player).getMoney(); will(returnValue(20));` `  // lancer des des` `  exactly(2).of(dice).throwDice();` `  will(onConsecutiveCalls(returnValue(4), returnValue(3)));` `  // le joueur a gagne, la banque le credite de 2 * 5€` `  oneOf(bank).credit(player, 5*2);` `  oneOf(bank).getMoney(); will(returnValue(200 - (5*2)));` `} });` `// player 2 joue et gagne` `context.checking(new Expectations(){ {` `  // player2 a 30€` `  oneOf(player2).getMoney(); will(returnValue(30));` `  // lancer des des` `  exactly(2).of(dice).throwDice();` `  will(onConsecutiveCalls(returnValue(4), returnValue(3)));` `  // le joueur a gagne, la banque le credite de 2 * 5€` `  oneOf(bank).credit(player2, 5*2);` `  oneOf(bank).getMoney(); will(returnValue(190 - (5*2)));` `} });`

	`` // player 3 joue et perd context.checking(new Expectations(){ { // player3 a 40€ oneOf(player3).getMoney(); will(returnValue(40)); // lancer des des exactly(2).of(dice).throwDice(); will(onConsecutiveCalls(returnValue(4), returnValue(2))); // le joueur a perdu, la banque le debite de 5€ oneOf(bank).debit(player3, 5); oneOf(bank).getMoney(); will(returnValue(180 + 5)); } }); game = new Game(bank, dice, dice); game.addPlayer(player); game.addPlayer(player2); game.addPlayer(player3); game.play(5); context.assertIsSatisfied(); assertTrue(game.isOpen()); ```
Résultat attendu	Scenario vérifié. Les deux premiers gagnent. La banque les crédite donc en conséquence. Le dernier joueur perd, il est donc débité. Le jeu reste ouvert.

Identifiant de test	testPlayersLose ()
Objectif	Jeu de test avec tous les joueurs qui perdent.
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player, player2
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player ``` // player 1 joue et perd context.checking(new Expectations(){ { // player1 a 20€ oneOf(player).getMoney(); will(returnValue(20)); // lancer des des exactly(2).of(dice).throwDice(); will(onConsecutiveCalls(returnValue(4), returnValue(1))); // le joueur a perdu, la banque le debite de 5€ oneOf(bank).debit(player, 5); oneOf(bank).getMoney(); will(returnValue(180 + 5)); } }); // player 2 joue et perd context.checking(new Expectations(){ { // player2 a 30€ oneOf(player2).getMoney(); will(returnValue(30)); ```

	`` // lancer des des exactly(2).of(dice).throwDice(); will(onConsecutiveCalls(returnValue(4), returnValue(2))); // le joueur a perdu, la banque le debite de 5€ oneOf(bank).debit(player2, 5); oneOf(bank).getMoney(); will(returnValue(185 + 5)); } }); game = new Game(bank, dice, dice); game.addPlayer(player); game.addPlayer(player2); game.play(5); context.assertIsSatisfied(); assertTrue(game.isOpen()); ```
Résultat attendu	Scenario vérifié. Tous les deux joueurs ont perdu. La banque les débite donc. Le jeu reste tout de même ouvert.
Identifiant de test	testAllPlayersWin ()
Objectif	Jeu de test avec tous les joueurs qui gagnent et la banque qui saute.
Environnement	JUnitRuleMockery context, Game game, IDice dice, IBank bank, IPlayer player, player2, player3
Etapes	Creation des mock pour IBank bank, IDice dice et IPlayer player ``` context.checking(new Expectations(){ { // player a 20€ oneOf(player).getMoney(); will(returnValue(20)); // lancer des des exactly(2).of(dice).throwDice(); will(onConsecutiveCalls(returnValue(4), returnValue(3))); // le joueur a gagne, la banque le credite de 2 * 20€ oneOf(bank).credit(player, 10*2); oneOf(bank).getMoney(); will(returnValue(50 - (10*2))); } }); context.checking(new Expectations(){ { // player a 20€ oneOf(player2).getMoney(); will(returnValue(20)); // lancer des des exactly(2).of(dice).throwDice(); will(onConsecutiveCalls(returnValue(4), returnValue(3))); // le joueur a gagne, la banque le credite de 2 * 20€ oneOf(bank).credit(player2, 10*2); oneOf(bank).getMoney(); will(returnValue(50 - (10*2))); } }); context.checking(new Expectations(){ { // player a 20€ oneOf(player3).getMoney(); will(returnValue(20)); ```

	`` // lancer des des exactly(2).of(dice).throwDice(); will(onConsecutiveCalls(returnValue(4), returnValue(3))); // le joueur a gagne, la banque le credite de 2 * 20€ oneOf(bank).credit(player3, 10*2); oneOf(bank).getMoney(); will(returnValue(10 - (10*2))); } }); game = new Game(bank, dice, dice); game.addPlayer(player); game.addPlayer(player2); game.addPlayer(player3); game.play(10); context.assertIsSatisfied(); ```
Résultat attendu	Scenario vérifié. Tous les joueurs gagnent et la banque saute. Levée d'exception BankRuptException.