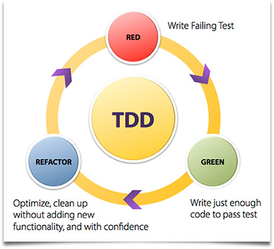
Initiation au TDD

Le principe



<http://codemanship.co.uk/parlezuml/blog/?postid=987>

Disclaimer

Pas de magie : de la pratique

<http://codemanship.co.uk/parlezuml/blog/?postid=1533>

Ordre des opérations et conventions

Nom de la classe de tests (Camel case) :

public class MyObjectShould {...

Nom des méthodes de tests : continuer une phrase commençant par "My object should ..." et décrire ce qui va être fait et testé :

public void do\_something\_cool() {...

Ecriture du corps de la méthode de tests dans cet ordre (de base en haut) :

* Vérification(s)

org.junit.Assert.[...]

* Action(s)

myObject.doSomethingCool();

* Initialisation(s)

MyObject myObject = new MyObject ();

Utiliser son IDE pour générer les classes et méthodes nécessaires à la compilation.

***Kata d'illustration : la stack.***

Enchaînement Red, Green, Refactor

Red : Partir d'exemple simplissimes.

Green : Faire une première implémentation très naïve (Au plus simple, en créant de la duplication).

Refactor : Faire apparaitre la duplication pour mieux la supprimer.

***Kata d'illustration : le roman converter.***

Utiliser le vocabulaire du métier

Le rapport d'exécution des tests doit absolument être lisible par le métier : reprendre des phrases de la spécifications et les tourner sous la forme en "[My object] should [do something cool]".

Le code du test doit être compréhensible par le métier :

* JUnit et les méthodes static de Assert sont là pour cela !
* Ne pas hésiter à faire des imports statiques.

Le code du test doit être le plus compréhensible possible, y compris par le métier :

* Reprendre les mots de la spécifications pour nommer ses variables
* Refactorer en sous fonctions ayant des noms explicites
* Coder "fluent" ( <https://martinfowler.com/bliki/FluentInterface.html> )

***Kata d'illustration : le Leap Year.***

Mocker pour régner

Faire des "new" sur des interfaces simples, créer des interfaces simples sur lesquels d'autres blocs de code viendront s'interfacer, vérifier que l'information est passée aux tiers (Collaborateurs) sans se soucier de ce qu'il doit en faire (C'est l'objet des tests sur ce collaborateur).

**Kit de survie avec les Mock**

|  |
| --- |
| public interface MyCollaborator;    // On peut créer des mocks sur des interfaces sans mockito :  MyCollaborator collaborator = new MyCollaborator () {};    // On peut créer des mocks sur toutes sortes d'objet :  MyCollaborator collaborator = Mockito.mock(MyCollaborator.class);    // On peut ajouter du comportement à un objet :  Mockito.when(collaborator.getName()).thenReturn("MyCollaborator");    // On peut vérifier qu'une méthode de l'objet à bien été appelée :  Mockito.verify(collaborator, Mockito.times(1)).getName(); |

<http://site.mockito.org/>

***Kata d'illustration : le PaymentService.***