

INF3143

Modélisation et spécification formelles des logiciels

Hiver 2018

Alexandre Terrasa

Département d'informatique, UQÀM



Tests unitaires avec JUnit

Rappel: test unitaire

Tester une **unité de code**:

- composant, module, classe, méthode

Vérifier un **comportement**:

- cas normaux
- cas limites
- cas anormaux

JUnit

Un framework de test unitaire pour Java

- K. Beck & E. Gamma (1998)

Intégré à la plupart des IDEs Java:

- BlueJ
- Eclipse
- NetBeans
- IntelliJ

Série xUnit

Port dans de nombreux langages:

- FlexUnit ActionScript
- NUnit C#
- HUnit Haskell
- JSUnit JavaScript
- PHPUnit PHP
- PyUnit Python
- ...

Terminologie JUnit

Cas de test (*test case*)

- test d'une méthode dans un contexte particulier

Suite de test (*test suite*)

- ensemble de cas de tests

Test unitaire (*unit test*)

- plus petite unité à tester
- en Java: la classe

Exemple

Création d'un test unitaire avec JUnit et
NetBeans

Écrire des tests unitaires

API et tests unitaires

API: Application Programming Interface

Ensemble de classes, méthodes ou fonctions permettant de manipuler un logiciel ou une bibliothèque

Comment tester les services d'une API?

Tester une méthode d'une API

Se mettre dans la peau du client

- Comment va-t-il utiliser la méthode?

Partir de la signature

- tester les arguments et l'absence d'arguments
- tester les valeurs de retour attendues
- tester les Exceptions attendues

Assertions JUnit

`fail(message)`

échoue toujours

`assertTrue(condition)`

échoue si la condition est fausse

`assertFalse(condition)`

échoue si la condition est vraie

`assertEquals(exp, actual)`

échoue si les deux objets ne sont pas égaux

Assertions JUnit

`assertNotNull(object)`

échoue si objet n'est pas nul

`assertNotNull(object)`

échoue si objet est nul

`assertSame(exp, actual)`

échoue si les deux instances sont différentes

`assertNotSame(exp, actual)`

échoue si les deux instances sont la même

Exemple

Utilisation des assertions

Annotations de test

`@Test`

identifie une méthode de test

`@Test(expected=Exception.class)`

échoue si le test ne lance pas l'exception attendue

`@Test(timeout=100)`

échoue si le test ne fini pas avant le timeout

Exemple

Utilisation des annotations
timeouts et exceptions

Exécution d'un test

1. **Initialisation** (Setup)
 - définir le contexte et l'environnement du test
2. **Exercice** (Trigger)
 - appel de l'unité à tester
3. **Vérification** (Verify)
 - vérification du résultat ou état produit
4. **Désactivation** (Teardown)
 - nettoyage, remettre le système dans son état initial

G. Meszaros, xUnit Test Patterns - Refactoring Test Code. Addison-Wesley, Upper Saddle River, NJ, 2007.

Annotations d'exécution

@Before, @After

identifie une méthode à exécuter avant (setup) / après (tear down) chaque test

@BeforeClass, @AfterClass

identifie une méthode à exécuter avant / après tous les tests d'une classe

@Ignore, @Ignore("Raison")

désactive un test

Exemples

Utilisation des annotations

@Before, @After

Où ranger les tests?

Dans **un répertoire différent des sources**:

- séparer tests et code réel
- faciliter la navigation dans le projet
- faciliter l'accès à la “documentation”

Dépend des standards utilisés:

- Netbeans `src/` `tests/`
- Maven `src/main/java/` `src/test/java/`

Tests unitaires et couverture de code

Tester quoi?

Plusieurs opinions sur le sujet

- absolument tout
- l'interface publique
- les morceaux complexes, critiques
- comportement vs. état

Règles générales

- plus on a de tests, moins on risque de régression
- pas besoin de tester le code trivial (getter / setter)

Couverture de code

Taux de code source exécuté par les tests

Couverture de code

Idée générale:

- on veut tester tous les chemins d'exécution possibles

⇒ S'assurer que tout le code écrit fonctionne

Calculer la couverture

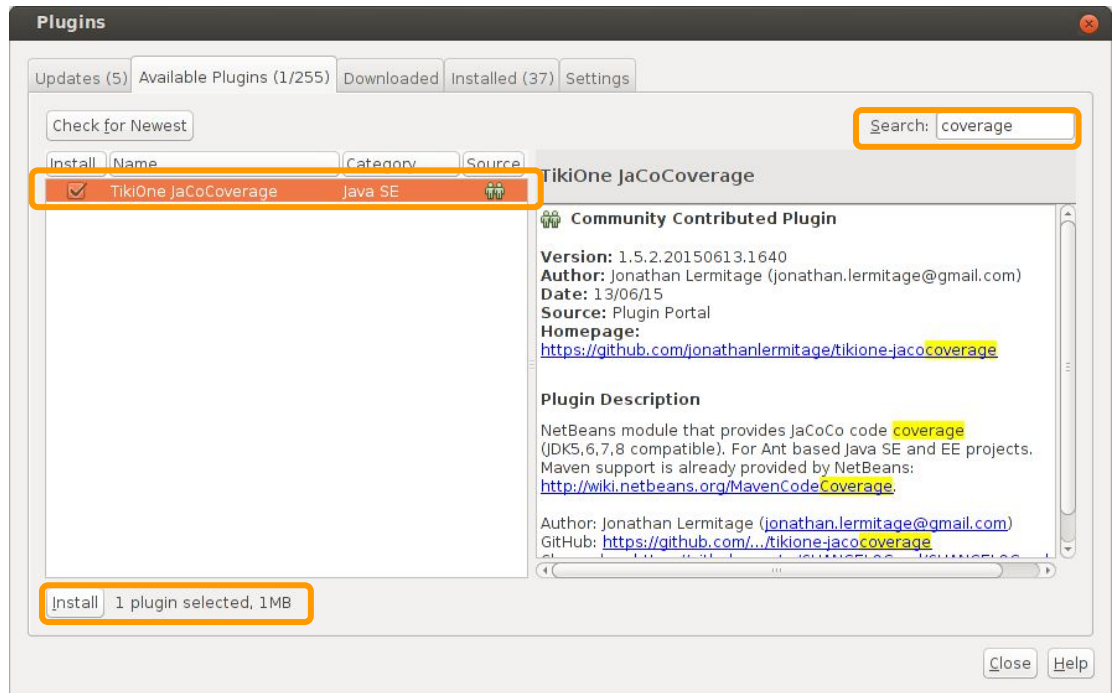
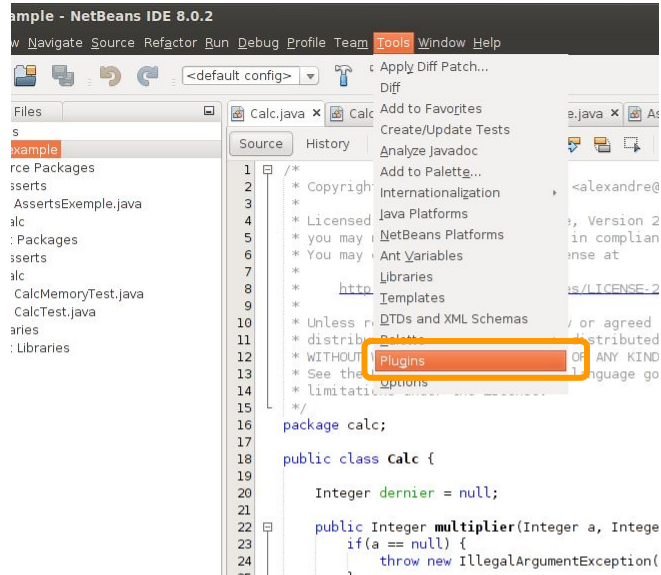
Il existe beaucoup d'outils pour le faire dans la plupart des langages.

Un exemple: [JavaCoCo](#)

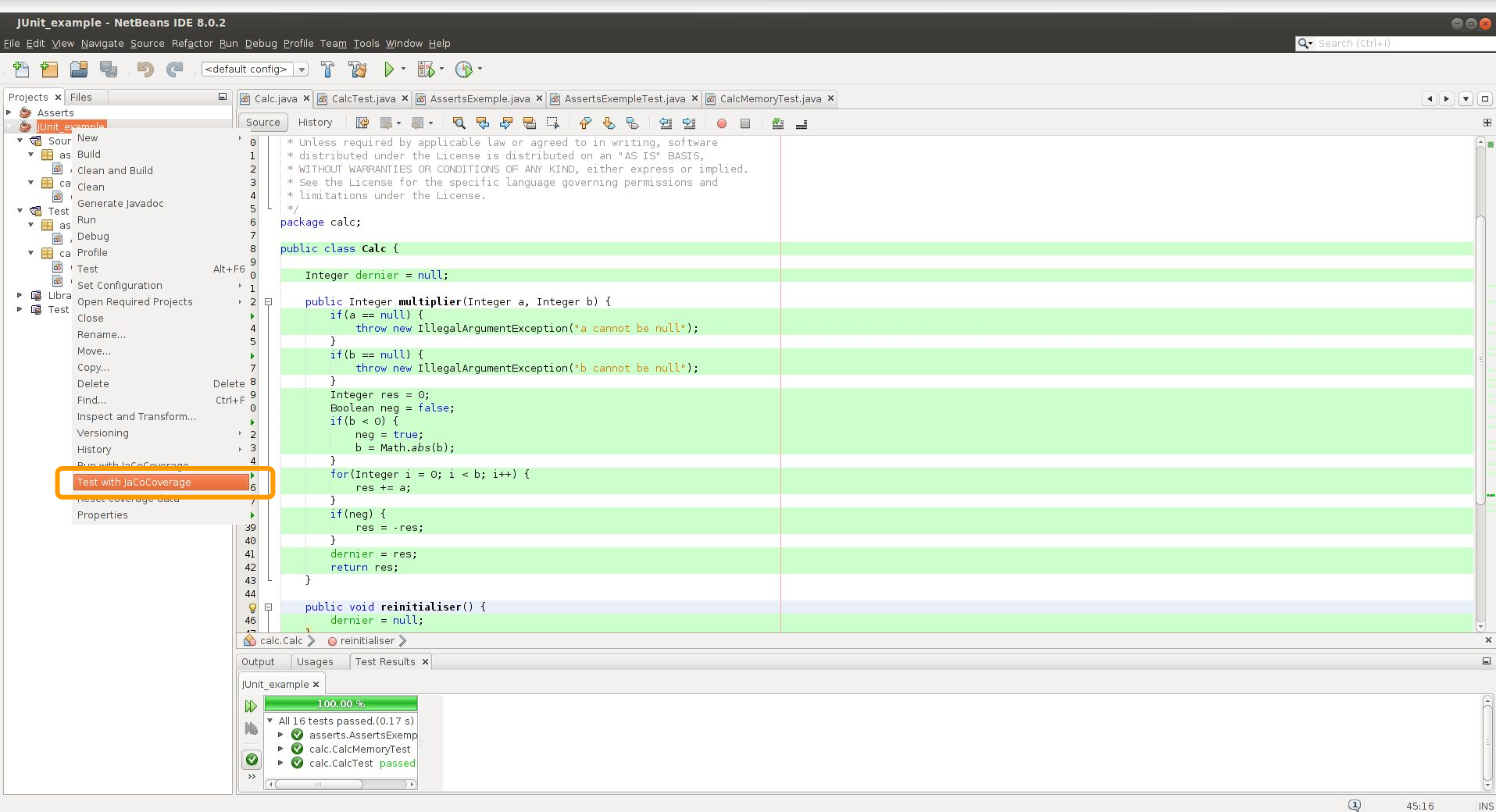
avec un plugin pour NetBeans

[TikiOne JavaCo Coverage](#)

Installation du plugin JavaCoco dans NetBeans



Utiliser JavaCoco dans NetBeans







Rapport de couverture

JaCoCoverage analysis of project "JUnit_example" (powered by JaCoCo from EcEmma) > asserts > AssertsExemple

 [Sessions](#)

AssertsExemple

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getTrue()		100%		n/a	0	1	0	1	0	1
getFalse()		100%		n/a	0	1	0	1	0	1
getOne()		100%		n/a	0	1	0	1	0	1
getFoo()		100%		n/a	0	1	0	1	0	1
AssertsExemple()		0%		n/a	1	1	1	1	1	1
Total	3 of 14	79%	0 of 0	n/a	1	5	1	5	1	5

Created with [JaCoCo](#) 0.7.5.201505241946

Bonnes pratiques

Qualités d'un bon test

Précis

- teste un comportement en particulier
- un test = une assertion

Répétable et déterministe

- s'attend toujours au même comportement

Isolé et indépendant

- ne dépend de rien d'autre

Exemples

Écrire des tests répétables

Nommer les tests

Pour les classes:

- suffixer avec “Test”

Pour les méthodes:

- décrire le contexte et le résultat attendu
- Approche Sujet_Scénario_Résultat

`ProductPurchaseAction_IfStockIsZero_RendersOutOfStockView()`

<http://blog.stevensanderson.com/2009/08/24/writing-great-unit-tests-best-and-worst-practises/>

Ne jamais faire confiance au client de vos services

Le client utilise mal vos services

- mauvais inputs, valeurs limites
arguments nuls, valeurs inattendues
- comportements limites
pop() sur une collection vide

Valeurs anormales et limites

Sur les entiers:

- entiers négatifs
- zéro
- maximums négatifs et positifs
- null si accepté
- bornes logiques s'il y a lieu
- ...

<https://blogs.msdn.microsoft.com/jledgard/2003/11/03/software-testing-6-good-tests-for-bad-parameters/>

Valeurs anormales et limites

Sur les chaînes:

- null String
- Chaîne vide
- ASCII, Unicode
- Délimiteurs de fin de chaîne
- Espaces, sauts de lignes, tabulations
- ...

Valeurs anormales et limites

Sur les fichiers:

- fichier inexistant
- fichier corrompu
- fichier contenant de mauvaises données
- mauvais format
- différents encodages (ASCII, UTF-8...)
- chemin contenant des caractères inattendus
- ...

Tester les exceptions

```
@Test(expected=MyException.class)
```

1. Lister les exceptions qui doivent être levées
2. Écrire un test pour chaque exception
3. Vérifier que l'exception est bien levée et que le message est clair, compréhensible

Maintenance des tests

Quand on trouve un nouveau bogue:

1. Écrire un test qui montre comment reproduire le bogue
2. S'assurer que le test échoue
3. Corriger la source du bogue dans le code
4. Faire passer le test

⇒ S'assurer de ne pas reproduire le bogue

Tests unitaires

Usages avancés

Tester avec des services externes

Certains composants sont difficiles à tester

- connexion à une base de données
- connexion réseau
- ...

⇒ Comment écrire un test unitaire dans ce cas?

Objets factices

Un **objet factice** (*mock object*) simule le comportement d'un objet réel impossible à utiliser dans un test.

Le *mock* **présente la même interface** que l'objet qu'il simule:

- le client ignore qu'il interagit avec un mock au lieu de l'objet réel

Examples

Example Mocking

Test Driven Development

Développement piloté par les tests (TDD)

- écrire le test unitaire avec d'écrire le code source

Intérêts

- code toujours testé
- meilleure interface
- documentation gratuite

TDD Processus

Cycle préconisé de TDD

1. Écrire le test unitaire
2. Vérifier qu'il échoue
3. Écrire le code suffisant pour passer le test
4. Vérifier que le test passe
5. Nettoyer le code écrit

Examples

Exemple TDD

Commentaires et documentation

Les tests unitaires forment de la documentation

- mode d'emploi des services
- instantiation de classes
- appels de méthodes
- toujours à jour avec le code

Peut-on aller plus loin?

Des tests unitaires dans la documentation

Idée:

- intégrer des tests unitaires dans la documentation des services
- directement dans les commentaires du code
- donner un exemple d'usage et gagner un test unitaire gratuitement
- garder les commentaires synchronisés avec le code

Approche DocTest

Mécanisme partagé par plusieurs langages

- Python doctest

<https://docs.python.org/2/library/doctest.html>

- Go testable examples

<https://blog.golang.org/examples>

- Rust documentation as tests

<https://doc.rust-lang.org/nightly/book/documentation.html#documentation-as-tests>

- Nit Doc Units

<http://nitlanguage.org/tools/nitunit.html>

Exemple avec Nit DocUnits

```
class Calc
  # Multiply `a` by `b`.
  #
  # ~~~
  # var calc = new Calc
  # assert calc.multiply(1, -10) == -10
  # assert calc.multiply(10, 10) == 100
  # ~~~
  fun multiply(a, b: Int): Int do
    return a * b
  end
end
```


Exemples

Exemple avec les DocUnits Nit

Conclusion

Écrire des tests unitaires pour

- valider le comportement du code
- éviter les régressions
- vérifier l'utilisabilité des APIs

Qualités d'un bon test unitaire

- précision, répétabilité, indépendance

Ressources

- http://intra.info.uqam.ca/Members/mili_h/Enseignement/inf3140-aut10/notes-cours/assertions-tests-texte.pdf
- <http://www.vogella.com/tutorials/JUnit/article.html>
- VC HUNT, A. and THOMAS D. -- Pragmatic Unit Testing In Java with JUnit. -- The Pragmatic Bookshelf, Raleigh, NC, 2003.
- VC MESZAROS, G. -- xUnit Test Patterns—Refactoring Test Code. -- Addison-Wesley, Upper Saddle River, NJ, 2007.