

Outils de Gestion de Projet : Junit

- Nauwynck Arnaud
- arnaud.nauwynck@sgam.com
arnaud.nauwynck@gmail.com
- Montreuil le 28 mars 2008

Plan

- Introduction
- Quick Start Guide : 1 minute overview
- Eclipse & Maven : 5 minutes Démo
- Tests Unitaires vs Tests d'Intégration
- Ecrire des Tests : IOC, Mock-Objects
- Stratégie d'Assurance Qualité de Projet

What : Qu'est-ce que Junit ?

- J = C'est une librairie en Java
- Unit = Elle permet d'écrire des tests unitaires
- Et...
 - Il existe de nombreux portages dans d'autres technologies: NUnit (dotnet), CppUnit (C++), DbUnit, HttpUnit, ...

Why : Junit pourquoi ?

- Rempli un besoin fondamental :
L'assurance Qualité par le Test des programmes
- C'est un standard industriel
- Intégré dans la plupart des outils de projets

Who : Les Auteurs de Junit

- Erich Gama
 - Auteur principal du livre incontournable Gof : « Design Pattern »
 - Architecte du projet Eclipse
- Kent Beck
 - Auteur de eXtreme Programming, et de nombreux livres

When : Origines de Junit

- Les principes fondamentaux de Junit ont été entièrement désignés en qq heures, dans l'avion qui emmenait K. Beck et E. Gamma en conférence
- Pratiquement tout le code a été écrit en one-shot

Conséquences de son Origine

- La librairie est petite
- Le code est strictement adapté au besoin, ce n'est pas un framework usine à gaz!
- Le code est très homogène
- Le code a peu évolué ensuite (sauf pour les annotations de java 5)

Commentaire A Posteriori des Auteurs

Jamais dans l'histoire, un si petit bout de code n'a autant fait parler de lui et changer si profondément la vision de l'informatique

Plan

- Introduction
- Quick Start Guide : 1 minute overview
- Eclipse & Maven : 5 minutes Démo
- Tests Unitaires vs Tests d'Intégration
- Ecrire des Tests : IOC, Mock-Objects
- Stratégie d'Assurance Qualité de Projet

Quick Start: a 1 mn Overview

```
import junit.framework.TestCase;

public class MyTest extends TestCase {

    public MyTest(String name) {
        super(name);
    }

    public void test1() {
        assertTrue(true);
    }

    public void testOther() {
        assertEquals(2, 1+1);
    }
}
```

Explication

- On étend d'une classe Test, par défaut « **TestCase** »
 - La convention de nommage veut donc que l'on suffixe sa classe par « Test » (anglais=postfixé)
- Toutes les méthodes qui commencent par «**public void test** » seront invoquées
- On peut utiliser les méthodes utilitaires d'**assert**()** de la super classe

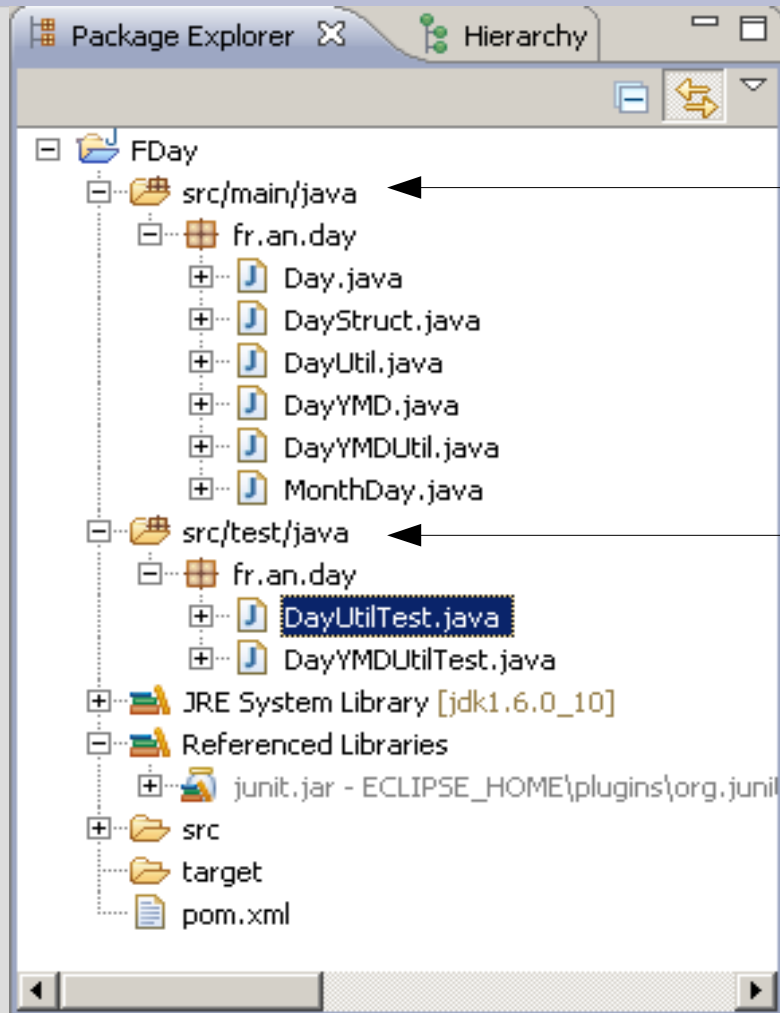
Exécution du Test

- De nombreux environnement savent s'intégrer avec ces classes de **junit.jar**
- **Il n'y a rien à faire d'autre!**
- Par exemple :
 - Sous Eclipse (Right-Click > Run As Junit Test)
 - Sous maven : « mvn install », « mvn test »
 - Sous maven : rapport de sites html ...

Plan

- Introduction
- Quick Start Guide : 1 minute overview
- Eclipse & Maven : 5 minutes Démo
- Tests Unitaires vs Tests d'Intégration
- Ecrire des Tests : IOC, Mock-Objects
- Stratégie d'Assurance Qualité de Projet

Configuration Tests sous Eclipse



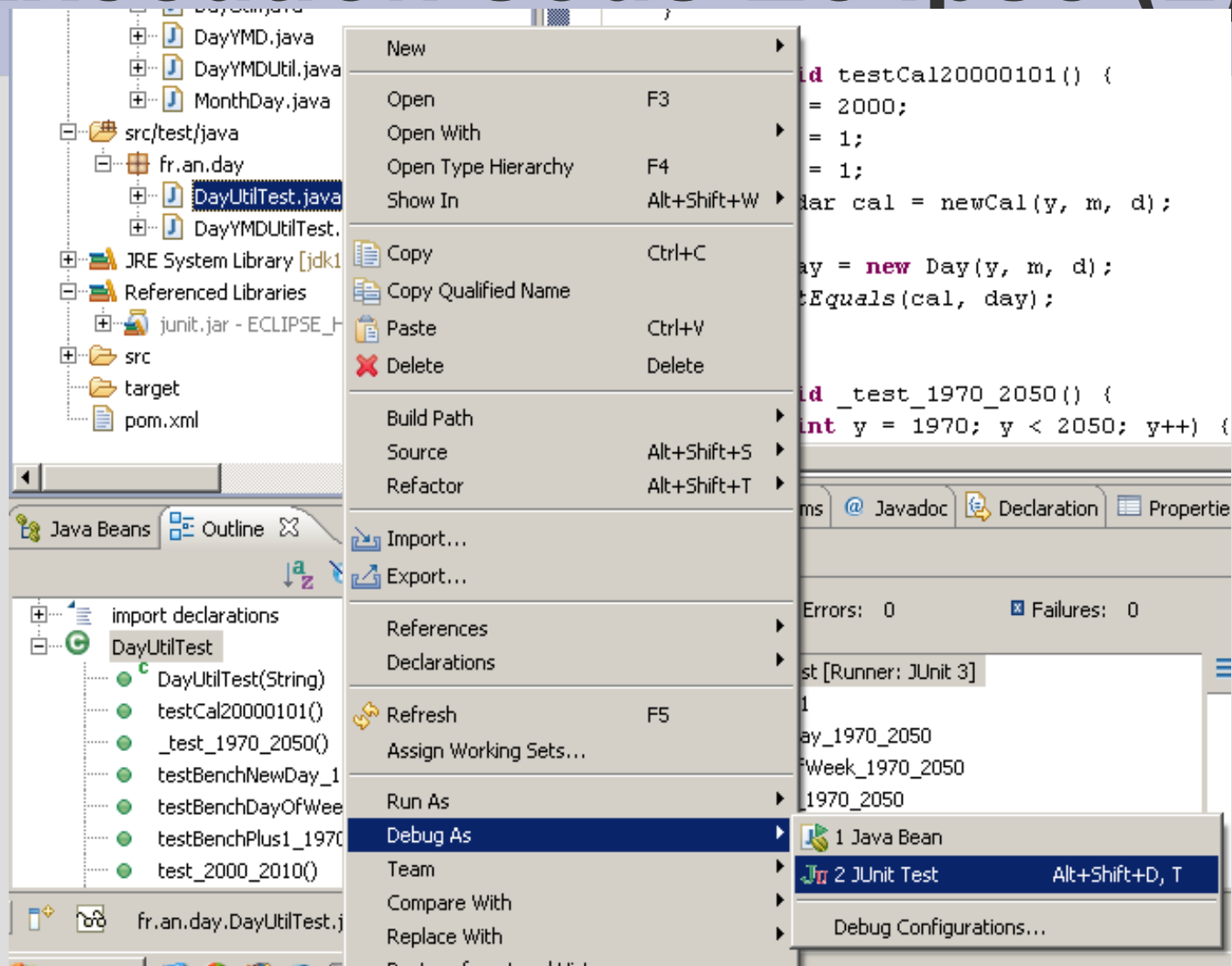
Répertoire des sources (src/main/java)

Répertoire des tests (src/test/java)

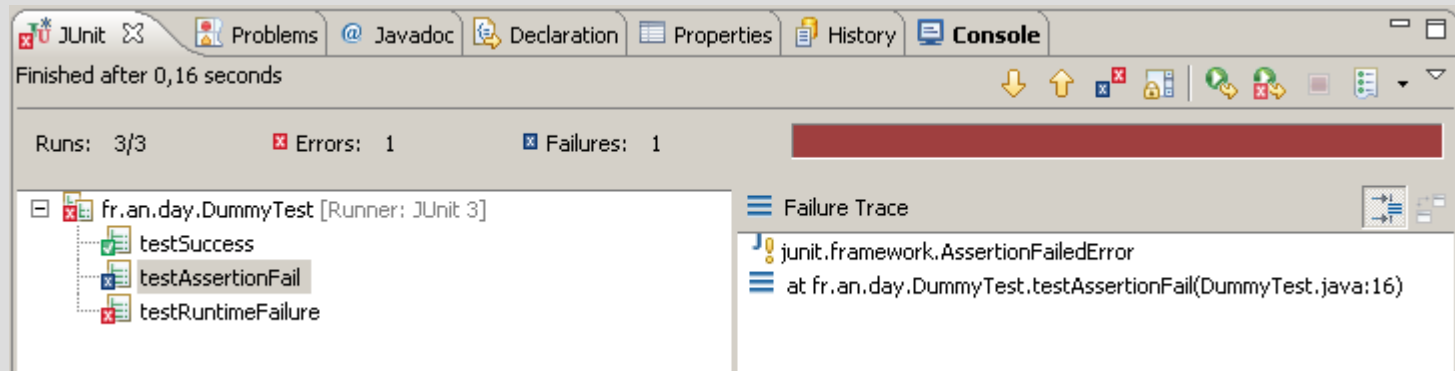
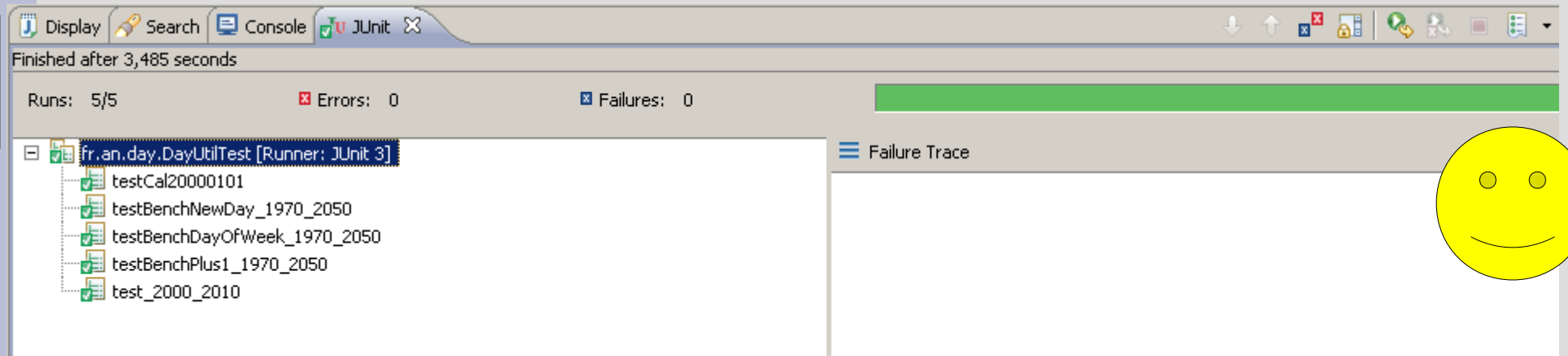
Librairie des tests (junit.jar)

(fourni dans Eclipse ou dans Maven..)

Exécution sous Eclipse (2)



Résultats sous Eclipse



Types de Résultats : Success/Failure/Error

```
DummyTest.java X

public class DummyTest extends TestCase {

    public DummyTest(String name) {
        super(name);
    }

    public void testSuccess() {
        assertTrue(true);
    }

    public void testAssertionFail() {
        assertTrue(false);
    }

    public void testRuntimeFailure() {
        throw new RuntimeException("failed...")
    }

}
```

JUnit Problems Javadoc Declaration P

Finished after 0,16 seconds

Runs: 3/3 Errors: 1 Failures: 1

fr.an.day.DummyTest [Runner: JUnit 3]

- testSuccess
- testAssertionFail
- testRuntimeFailure

Exécution sous Maven

- Par Défaut, les tests sont activés sous maven
- Les tests Unitaires sont exécutés à CHAQUE compilation !!!
 - Il est parfois intéressant de désactiver certains tests pour la rapidité du build :(!!!
- Si les tests ne passent pas... la compilation échoue !!

Test Passed <= Build Success

Test Failed => Build Failed

```
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.804 sec
```

```
Results :
```

```
Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
```

```
[INFO] [jar:jar]
```

```
[INFO] Building jar: C:\users\arnaud\devPerso\FDay\target\FDay-1.0-SNAPSHOT.jar
```

```
[INFO] [install:install]
```

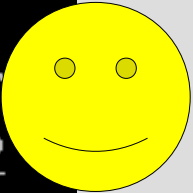
```
[INFO] Installing C:\users\arnaud\devPerso\FDay\target\FDay-1.0-SNAPSHOT.jar to
```

```
[INFO] -----
```

```
[INFO] BUILD SUCCESSFUL
```

```
[INFO] -----
```

```
[INFO] Total time: 13 seconds
```



```
Results :
```

```
Failed tests:
```

```
testAssertionFail(fr.an.day.DummyTest)
```

```
Tests in error:
```

```
testRuntimeFailure(fr.an.day.DummyTest)
```

```
Tests run: 12, Failures: 1, Errors: 1, Skipped: 0
```

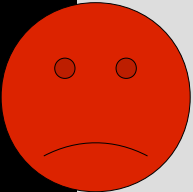
```
[INFO] -----
```

```
[ERROR] BUILD FAILURE
```

```
[INFO] -----
```

```
[INFO] There are test failures.
```

```
[INFO] -----
```



Mvn install mvn test

- Mvn install
 - Command de build standard =
compile les classe java + les classes de test
+ exécute les test
+ fabrique les jar + copie les jar dans le repo
- Mvn test
 - Execute les tests uniquement

Mvn install

```
C:\users\arnaud\devPerso\FDay>mvn install
[INFO] Scanning for projects...
[INFO] -----
[INFO] Building Fast Day Library
[INFO]   task-segment: [install]
[INFO] -----
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:compile]
[INFO] Compiling 6 source files to C:\users\arnaud\devPerso\FDay\target\classes
[INFO] [resources:testResources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Compiling 3 source files to C:\users\arnaud\devPerso\FDay\target\test-classes
[INFO] [surefire:test]
[INFO] Surefire report directory: C:\users\arnaud\devPerso\FDay\target\surefire-reports

-----
T E S T S
-----
Running fr.an.day.DummyTest
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.09 sec
Running fr.an.day.DayYMDUtilTest
Time to execute newDay (29220 days): 342.23134839151265 ms/1M
Time to execute dayOfWeek (29220 days): 1026.694045174538 ms/1M
Time to execute plus1 (29220 days): 684.4626967830253 ms/1M
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.684 sec
Running fr.an.day.DayUtilTest
Time to execute newDay (29220 days): 684.4626967830253 ms/1M
Time to execute dayOfWeek (29220 days): 684.4626967830253 ms/1M
Time to execute plus1 (29220 days): 684.4626967830253 ms/1M
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.814 sec

Results :

Tests run: 10, Failures: 0, Errors: 0, Skipped: 0

[INFO] [jar:jar]
[INFO] Building jar: C:\users\arnaud\devPerso\FDay\target\FDay-1.0-SNAPSHOT.jar
[INFO] [install:install]
[INFO] Installing C:\users\arnaud\devPerso\FDay\target\FDay-1.0-SNAPSHOT.jar to d:\Java\
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 17 seconds
[INFO] Finished at: Mon Jan 14 22:42:49 GMT+01:00 2008
[INFO] Final Memory: 5M/18M
[INFO] -----
```

Compilation classes

Compilation classes
tests JUnit

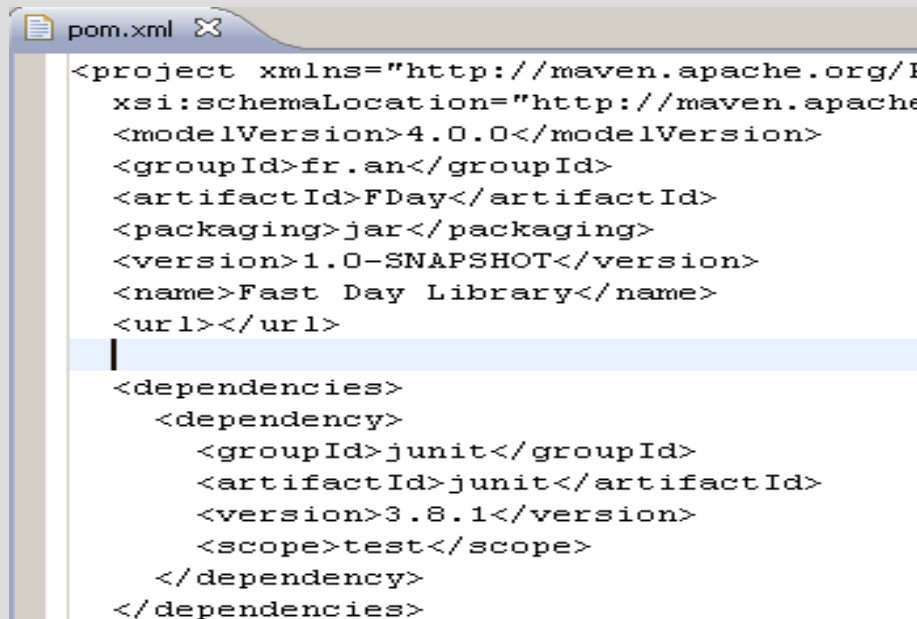
Execution
des tests JUnit

Build Jar

Copy Jar to Repo

Tests Unitaires sous Maven

- Un projet maven vide contient des tests...
cf « mvn archetype:create
-DgroupId=fr.iut.tp -DartifactId=tpJUnit »



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
  >
  <modelVersion>4.0.0</modelVersion>
  <groupId>fr.an</groupId>
  <artifactId>FDay</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>
  <name>Fast Day Library</name>
  <url></url>

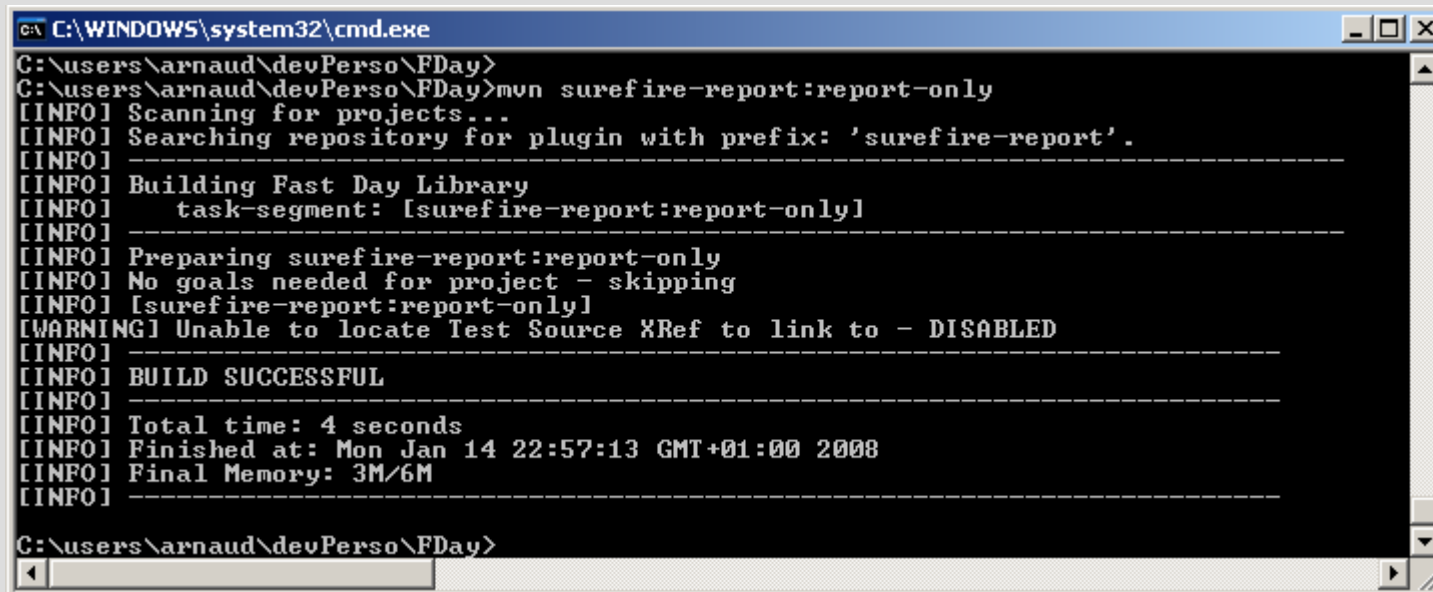
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
</project>
```

Dépendance pour junit.jar
explicite

... le plugin test est implicite

Rapport Site HTML sous Maven

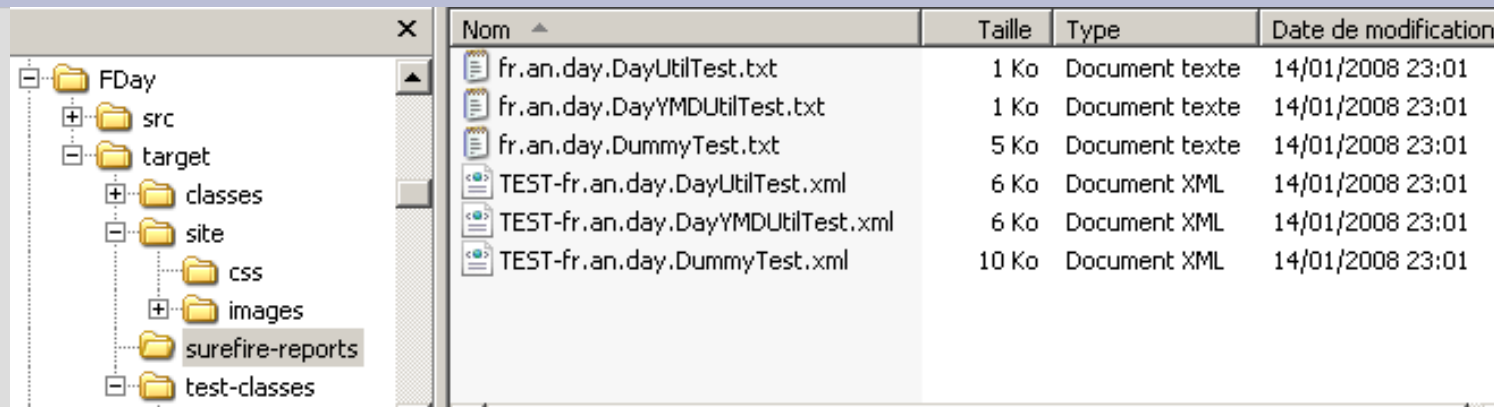
- Surefire est le plugin d'execution de Junit...
 - Mvn surefire-report:report ... run + generate
 - Mvn surefire-report:report-only ... only generate



```
C:\WINDOWS\system32\cmd.exe
C:\users\arnaud\devPerso\FDay>
C:\users\arnaud\devPerso\FDay>mvn surefire-report:report-only
[INFO] Scanning for projects...
[INFO] Searching repository for plugin with prefix: 'surefire-report'.
[INFO]
[INFO] Building Fast Day Library
[INFO] task-segment: [surefire-report:report-only]
[INFO]
[INFO] Preparing surefire-report:report-only
[INFO] No goals needed for project - skipping
[INFO] [surefire-report:report-only]
[WARNING] Unable to locate Test Source XRef to link to - DISABLED
[INFO]
[INFO] BUILD SUCCESSFUL
[INFO]
[INFO] Total time: 4 seconds
[INFO] Finished at: Mon Jan 14 22:57:13 GMT+01:00 2008
[INFO] Final Memory: 3M/6M
[INFO]
C:\users\arnaud\devPerso\FDay>
```

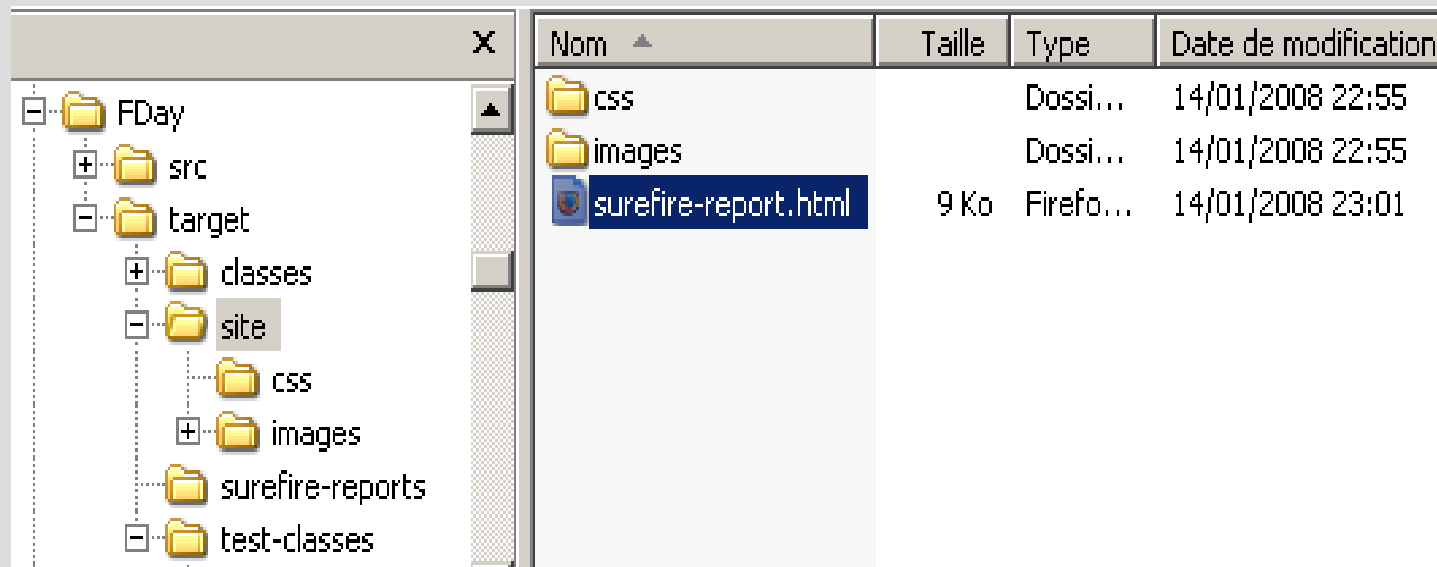
SureFire Reports

Txt / Xml / Html



The screenshot shows an IDE window with a file explorer on the left and a list of files on the right. The file explorer shows a project structure with folders: FDay, src, target, classes, site, css, images, surefire-reports, and test-classes. The list of files on the right shows the following data:

Nom	Taille	Type	Date de modification
fr.an.day.DayUtilTest.txt	1 Ko	Document texte	14/01/2008 23:01
fr.an.day.DayYMDUtilTest.txt	1 Ko	Document texte	14/01/2008 23:01
fr.an.day.DummyTest.txt	5 Ko	Document texte	14/01/2008 23:01
TEST-fr.an.day.DayUtilTest.xml	6 Ko	Document XML	14/01/2008 23:01
TEST-fr.an.day.DayYMDUtilTest.xml	6 Ko	Document XML	14/01/2008 23:01
TEST-fr.an.day.DummyTest.xml	10 Ko	Document XML	14/01/2008 23:01



The screenshot shows an IDE window with a file explorer on the left and a list of files on the right. The file explorer shows a project structure with folders: FDay, src, target, classes, site, css, images, surefire-reports, and test-classes. The list of files on the right shows the following data:

Nom	Taille	Type	Date de modification
css		Dossi...	14/01/2008 22:55
images		Dossi...	14/01/2008 22:55
surefire-report.html	9 Ko	Firefo...	14/01/2008 23:01

SureFire Report Result

Last Published: Mon Jan 14 22:57:13 GMT+01:00 2008

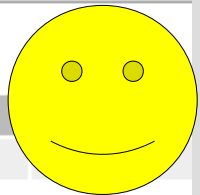
Project Documentation
Maven Surefire Report



Summary

[\[Summary\]](#)[\[Package List\]](#)[\[Test Cases\]](#)

Tests	Errors	Failures	Skipped	Success Rate
10	0	0	0	100%



Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

[\[Summary\]](#)[\[Package List\]](#)[\[Test Cases\]](#)

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
fr.an.day	10	0	0	0	100%	7

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

fr.an.day

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
	DummyTest	1	0	0	0	100%	0

SureFire Reports

Project Documentation
Maven Surefire Report



Summary

[\[Summary\]](#)[\[Package List\]](#)[\[Test Cases\]](#)

Tests	Errors	Failures	Skipped	Success Rate	Time
12	1	1	0	83.333%	4

Note: failures are anticipated and checked for with assertions while errors are unanticipated.

Package List

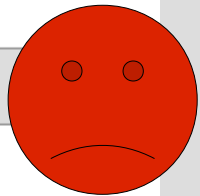
[\[Summary\]](#)[\[Package List\]](#)[\[Test Cases\]](#)

Package	Tests	Errors	Failures	Skipped	Success Rate	Time
fr.an.day	12	1	1	0	83.333%	4

Note: package statistics are not computed recursively, they only sum up all of its testsuites numbers.

fr.an.day

	Class	Tests	Errors	Failures	Skipped	Success Rate	Time
✖	DummyTest	3	1	1	0	33.333%	0
✔	DayYMDUtilTest	4	0	0	0	100%	2
✔	DayUtilTest	5	0	0	0	100%	2



SureFire Reports : Errors Details

DummyTest

	testSuccess	0
	testAssertionFail - [Detail]	0

```
junit.framework.AssertionFailedError
at junit.framework.Assert.fail(Assert.java:47)
at junit.framework.Assert.assertTrue(Assert.java:20)
at junit.framework.Assert.assertTrue(Assert.java:27)
at fr.an.day.DummyTest.testAssertionFail(DummyTest.java:16)
```

	testRuntimeFailure - [Detail]	0
	failed...	

```
java.lang.RuntimeException: failed...
at fr.an.day.DummyTest.testRuntimeFailure(DummyTest.java:20)
```

DayYMDUtilTest

	testBenchNewDay 1970 2050	0
---	---------------------------	---

Couverture de Tests par Code Coverage

- Avoir des tests ne suffit pas:
 - Les tests peuvent être bidons
 - Les tests peuvent oublier de couvrir certaines fonctionnalités
 - Les tests peuvent oublier de couvrir certains cas particulier (race condition / NPE / valeur 0)
- Sur un Projet il FAUT 2 métriques:
 - 100 % test success x 100% test coverage

Fonctionnement d'un Outil de Code Coverage

- Instrumentation
 - On instrumente un programme en rajoutant à chaque ligne de code, un marqueur «alreadySeenLineXXX = true »
- On exécute le programme
- On récupère tous les marqueurs
- On génère un rapport (html...)

Maven + Cobertura

- Cobertura = Couverture en Espagnol
- « Mvn cobertura:cobertura »
- cf. <http://mojo.codehaus.org/cobertura-maven-plugin/>

```

C:\WINDOWS\system32\cmd.exe
Instrument time: 22ms
[INFO] Instrumentation was successful.
[INFO] [resources:resources]
[INFO] Using default encoding to copy filtered resources.
[INFO] [compiler:testCompile]
[INFO] Nothing to compile - all classes are up to date
[INFO] [surefire:test]
[INFO] Surefire report directory: C:\users\arnaud\dev\Perso\FDay\target\surefire-reports

TESTS
Running fr.an.day.DummyTest
Tests run: 3, Failures: 1, Errors: 1, Skipped: 0, Time elapsed: 0.16 sec <<< FAILURE!
Running fr.an.day.DayUtilTest
Time to execute newDay <29220 days>: 4106.776189698152 ms/IM
Time to execute dayOfWeek <29220 days>: 2122.566235112938 ms/IM
Time to execute plus1 <29220 days>: 5 19883.641341546885 ms/IM
Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 86.314 sec
Running fr.an.day.DayUtilTest
Time to execute newDay <29220 days>: 10995.626283367557 ms/IM
Time to execute dayOfWeek <29220 days>: 37337.3401897514035 ms/IM
Time to execute plus1 <29220 days>: 5 18856.7422637235 ms/IM
Tests run: 5, Failures: 2, Errors: 0, Skipped: 0, Time elapsed: 2.023 sec <<< FAILURE!
Results: 2
Failed tests:
  testSessionFail(fr.an.day.DummyTest)
  testCallSessionFail(fr.an.day.DummyTest)
  test_2000_2010(fr.an.day.DayUtilTest)
Tests in error:
  testRunTimeFailure(fr.an.day.DummyTest)
Tests run: 12, Failures: 3, Errors: 1, Skipped: 0
Cobertura: Loaded information on 7 classes.
Cobertura: Saved information on 7 classes.
[ERROR] There are test failures.
[INFO] Cobertura 1.9 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file
Cobertura: Loaded information on 7 classes
Cobertura: INFO [main] net.sourceforge.cobertura.reporting.ComplexityCalculator - Can
Report time: 149ms
[INFO] Cobertura Report generation was successful.
[INFO] BUILD SUCCESSFUL
[INFO] Total time: 1 minute 41 seconds
[INFO] Finished at: Tue Jan 15 01:05:53 GMT+01:00 2008
[INFO] Final Memory: 4M/7M
[INFO]

```

Instrumentation des Tests

Exécution des Tests instrumentés

Génération du report Cobertura

Résultats de Mvn Cobertura

The image shows a file explorer on the left and a web browser displaying a Cobertura coverage report on the right.

File Explorer:

- Root: FDay
 - src
 - target
 - classes
 - cobertura
 - site
 - cobertura
 - css
 - images
 - surefire-reports
 - test-classes

Browser: Coverage Report - Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///C:/users/arnaud/devPerso/FDay/target/site/c

Coverage Report - All Packages

Package	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	7	56 % 203/364	42 % 62/148	2,417
fr.an.day	7	56 % 203/364	42 % 62/148	2,417

Report generated by [Cobertura](#) 1.9 on 15/01/08 01:05.

Packages

- [All](#)
- [fr.an.day](#)

All Packages

Classes

- [Day](#) (24 %)
- [DayStruct](#) (0 %)
- [DayUtil](#) (63 %)
- [DayYMD](#) (24 %)
- [DayYMDUtil](#) (96 %)
- [DummyTest](#) (0 %)
- [MonthDay](#) (80 %)

Done

Résultats de Cobertura (Détail)

Coverage Report - Mozilla Firefox

File Edit View History Bookmarks Tools Help

file:///C:/users/arnaud/devPerso/FDay/target/site/cobertura/inde: Google

Coverage Report - fr.an.day.DayUtil

Classes in this File	Line Coverage	Branch Coverage	Complexity
DayUtil	63 % 87/138	44 % 28/64	2,4

Line	Class	Line Coverage	Branch Coverage	Complexity	Code
1	package fr.an.day;				<code>package fr.an.day;</code>
2		213 0			<code>res = firstDayOfYear[offsetToYear] - firstDayOfY</code>
3	import	214			<code>} else {</code>
4		215 0			<code>throw new UnsupportedOperationException();</code>
5	0 public	216			<code>}</code>
6		217 0			<code>return res;</code>
7		218			<code>}</code>
8	2	219			<code>public static int getLastDayOfMonth(int year, int month) {</code>
9	2	220			<code>int res = maxDayOfMonth(month);</code>
10		221 11762			<code>if (month == 2 && isLeapYear(year)) {</code>
11		222 11762			<code>res = 29;</code>
12		223 246			<code>}</code>
13		224			<code>return res;</code>
14		225 11762			<code>}</code>
15		226			<code>public static int getLastDayCountOfYear(int year) {</code>
		227			<code>return (!isLeapYear(year)) ? 365 : 366;</code>
		228			<code>}</code>
		229 200			<code>public static boolean isLeapYear(int y) {</code>
		230			<code>int offset = y - YEAR_LEAP_ARRAY_OFFSET;</code>
		231			<code>if (y >= YEAR_LEAP_ARRAY_OFFSET && offset < YEAR_LEAP_AR</code>
		232			<code>return yearLeapArray[offset];</code>
		233 351828			<code>} else {</code>
		234 351828			<code>// should not occur...</code>
		235 351828			<code>return _slow_isLeapYear(y);</code>
		236			<code>}</code>
		237			<code>}</code>
		238 0			<code>}</code>
		239			<code>}</code>

All Packages

Classes

- Day (24 %)
- DayStruct (0 %)
- DayUtil (63 %)

Plan

- Introduction
- Quick Start Guide : 1 minute overview
- Eclipse & Maven : 5 minutes Démo
- Tests Unitaires vs Tests d'Intégration
- Ecrire des Tests : IOC, Mock-Objects
- Stratégie d'Assurance Qualité de Projet

Tests Dépendants de J2EE

- Rapidité
 - Un serveur d'application (weblogic, Jboss..) peut mettre longtemps à démarrer : > 3 minutes
 - On veut des tests rapides < 2 minutes!!!
 - On ne veut pas du serveur
- Dépendances
 - JNDI, EJB, JMS, JDBC ... sont des dépendances difficiles à tester

Définition Test Unitaires / Intégration

- Un test **Unitaire** est un test qui ne dépend que d'une seule classe prise unitairement
- Un test d'**Intégration** a besoin des composants/process démarrés (server, base de donnée, etc...)

Difficulté des Tests

« Code Spaggethi » / Classes Singleton

- A dépend de B et C, B dépend de C et D, ...
X dépend de la base de donnée, Y dépend du serveur weblogic
- Les dépendances ne se voient pas...tant que l'on a pas essayé de tester unitairement !!!
- Les singletons sont les pires dépendances

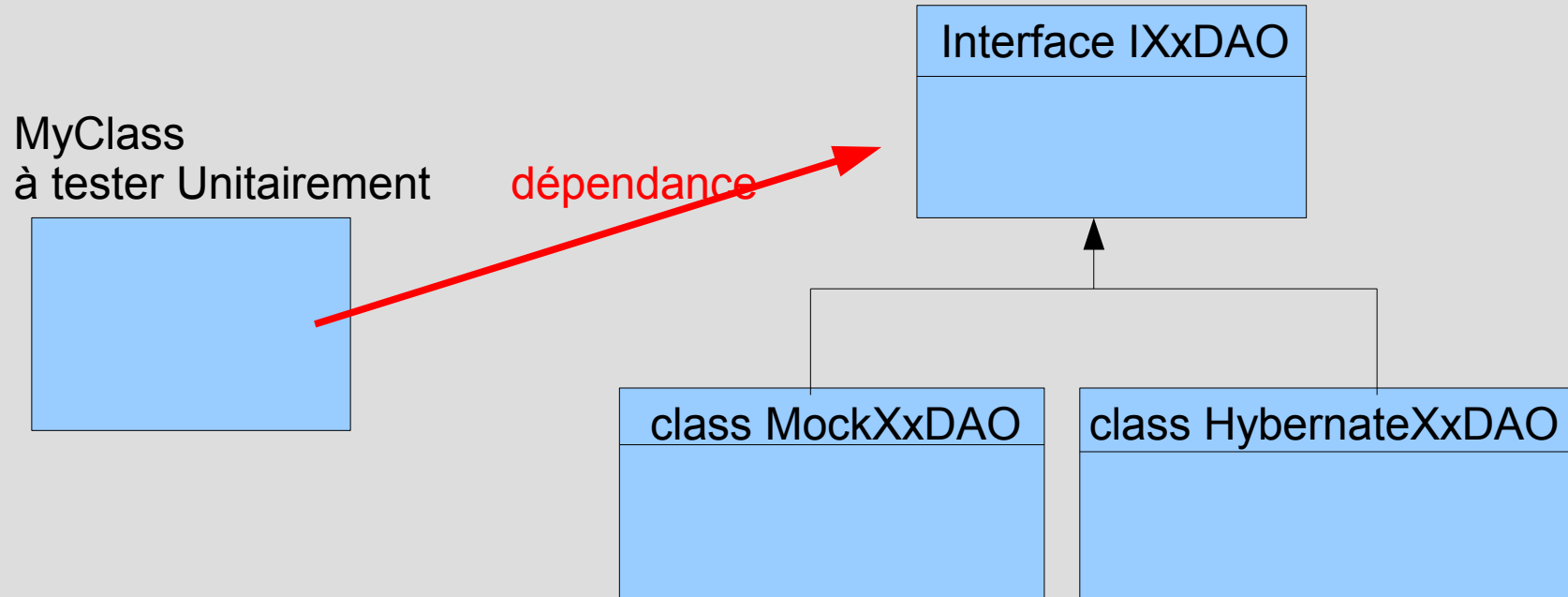
Plan

- Introduction
- Quick Start Guide : 1 minute overview
- Eclipse & Maven : 5 minutes Démo
- Tests Unitaires vs Tests d'Intégration
- Ecrire des Tests : IOC, Mock-Objects
- Stratégie d'Assurance Qualité de Projet

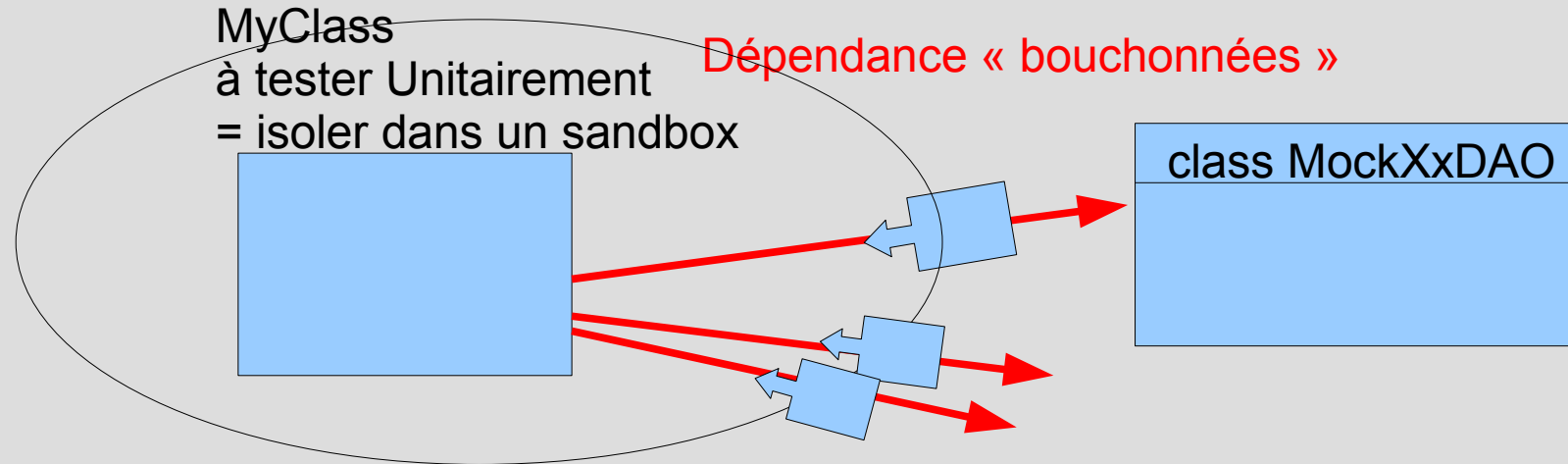
Utiliser les Mock-Objects

- **Mock** = objet bouchon, pour les tests seulement
- On s'en sert pour isoler la classe à tester
- Un Mock implémente la même interface que la classe de dépendance à tester, mais donne une implémentation, plus simple (bidon)

Mock Object + Interface

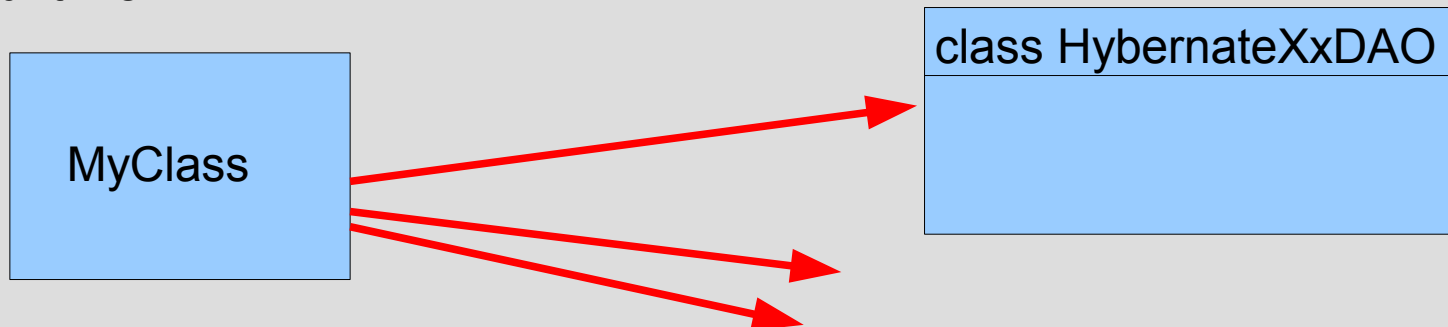


Config Rt=Default / Test=Mock



Config Test Unitaire

Config Au Runtime



Utiliser l'IOC

Config Rt=Default / test=Mock

- Problème
 - **Au runtime** = il faut utiliser la vraie classe
 - **Au test** = il faut utiliser le Mock Object à la place
 - On ne veut pas rajouter de « if(TEST) »!!!
- On utilise pour cela l'**IOC**
= Inversion Of Control
... synonym = **Dependency Injection**
- **Spring** est la librairie standard d'IOC

Exemple de Dépendance DAO

- Class MyClass {
 public void f() {
 HibernateXXDAO dao = **get singleton...**
 dao.create(); dao.update(); ...
 }
}
 - Class HibernateXXDAO extend Hibernate { // CRUD
 public X create(X obj) { hybernate.create(...) }
 public X findById(int id) { hybernate.findById(...) }
 public void update(X obj) { hybernate.update(...) }
 public void delete(X obj) { hybernate.delete(...) }
}
- 
- Dépendance !!!

Step 1 : Simplifier l'API de Dépendance

- Class MyClass {
 public void f() {
 IXXDAO dao = **get singleton...**
 dao.create(); dao.update(); ...
 }
}
 - Interface IXXDAO { // CRUD
 public X create(X obj);
 public X findById(int id);
 public void update(X obj);
 public void delete(X obj);
}
 - Class HybernateXxxDAO
 implements IxxDAO {
 public X create(X obj)
 { ... }
 ...
}
-
- ```
graph TD; MyClass -- "Dépendance Compilation Time" --> IXXDAO; MyClass -- "Dépendance RunTime !!!" --> HybernateXxxDAO; HybernateXxxDAO -- "implements" --> IXXDAO;
```

# Step 2 : Injecter les Dépendances

- 2 Méthodes :
  - Spring wiring = introduire un champ java, le configurer par Spring, par introspection (non intrusive)
  - Use Spring ApplicationContext API (intrusive)  
Idem singleton... mais l'implémentation est sortie dans un fichier de configuration Spring!

# IOC Méthode 1 (non intrusive)

- Class MyClass {  
    private IXXDAO dao;  
  
    // config par introspection+constructor  
    public MyClass(IXXDAO dao) {  
        this.dao = dao;  
    }  
    // ou par introspection+getter+setter ...  
  
    public void f() {  
        dao.create(); dao.update(); ...  
    }  
}

# IOC Méthode 2 (intrusive)

- Class MyClass {  
    public void f() {  
        ApplicationContext ctx = get App singleton...  
        IXXDAO dao = (IXXDAO) ctx.getBean(«xxDao»);  
        dao.create(); dao.update(); ...  
    }  
}
- On utilise souvent le full name de l'interface comme nom de bean:  
    IXXDAO dao = (IXXDAO) ctx.getBean(IXXDAO.class);

# Plan

- Introduction
- Quick Start Guide : 1 minute overview
- Eclipse & Maven : 5 minutes Démo
- Tests Unitaires vs Tests d'Intégration
- Ecrire des Tests
- Stratégie d'Assurance Qualité de Projet

# Réalité des Tests dans le Code Existant

- Souvent le code existant d'un projet n'est pas testé !!!!
- Impossible à refactorer ... impossible à maintenir sans peine
- Il faut écrire par petits bouts des tests sur l'ancien code
- et développer systématiquement des tests pour le nouveau code!!

# Si... yakafocon...

- Les tests constituent un problème difficile
- En partie parce que les décideurs ne l'intègre pas dans les budgets et plannings !!!
- C'est toute la mentalité de la gestion de projet qu'il faut revoir
- C'est nouveau (à peine 10 ans... !!!!)

# Les Tests en Pratique

- Les équipes sont scindés:
  - MOE (Maitrise d'Oeuvre = développeurs)
  - MOA (Maitrise d'Ouvrage = fonctionnels?)
  - QA = équipe Qualité / tests Winrunner...
  - Supports / Testeurs occasionnels
- Les tests constituent un problème transverse / long terme / non chiffré !!!!
- L'entreprise gère le court terme

# Métriques de Qualité

- Les outils comme Maven + Intégration Continue permettent d'avoir des métriques de qualité concrètes au jour le jour
  - Donc de détecter le changement
  - De détecter les points faibles potentiels
  - De consolider les points fragiles connus
  - De recetter (certifié) ce qui fonctionne
- Facile à mettre en place

# Les Approches Agiles

- Toutes les approches modernes (<5ans) consistent à augmenter l'Agilité
- Plus jamais de longs cycle en V
- Utilisation des outils d'Intégration Continue
- Tests (Unitaires) automatisés dans le processus !!  
... de la conception, dev (IOC) ... jusqu'au build et livraisons

# Junit : composant essentiel de eXtreme Programming

- Parmi les pratiques de eXtreme Programming (cf K. Beck), on trouve
  - Les tests Unitaires
  - Le refactoring
  - Le Pair-Programming, le Code Review
  - Le rapport Client-Fournisseur
  - La compréhension du coeur du métier (cf Model Driven Design)
  - ....

# Junit / eXtreme Programming

- Tous les concepts de eXtreme Programming sont liés
- Souvent, les détracteurs ne veulent pas appliquer telle ou telle partie...
  - Certains disent que le Pair-Programming fait perdre 50% de productivité (?!)
  - Certains disent que l'écriture des tests rallonge de 100% le temps de dev (?!)

# Agile / Junit / Extreme Programming

- Ce n'est pas un hasard si K. Beck, le co-concepteur de Junit est aussi l'auteur de eXtreme Programming!!!
- Sans aller jusqu'au bout de eXtreme Programming , il FAUT faire des tests Unitaires
- Sans Tests, point de salut

# Conclusion

- Questions ??
- Alors TP !
- Mettre en place Junit dans Eclipse + maven (Surefire + Copertura), par exemple pour un projet existant + Subversion !