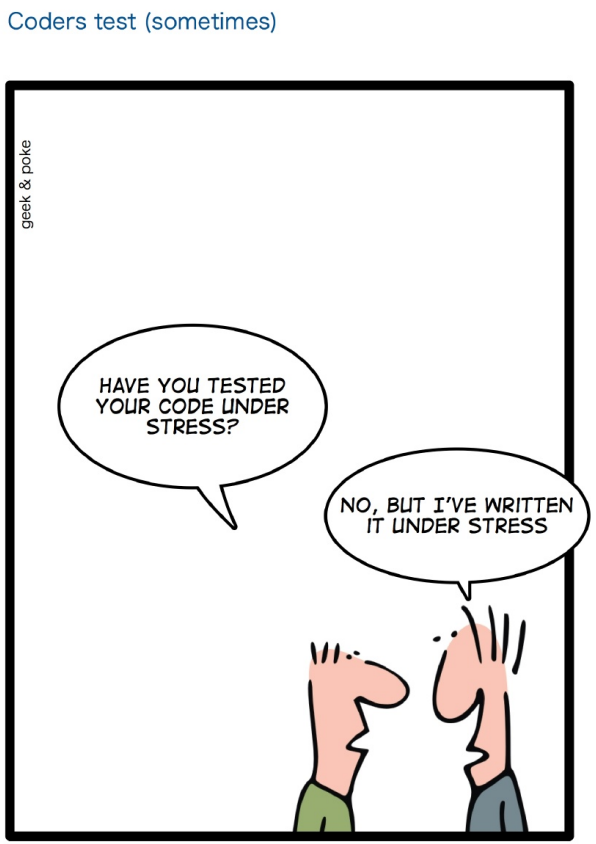
Tests unitaires avec JUnit

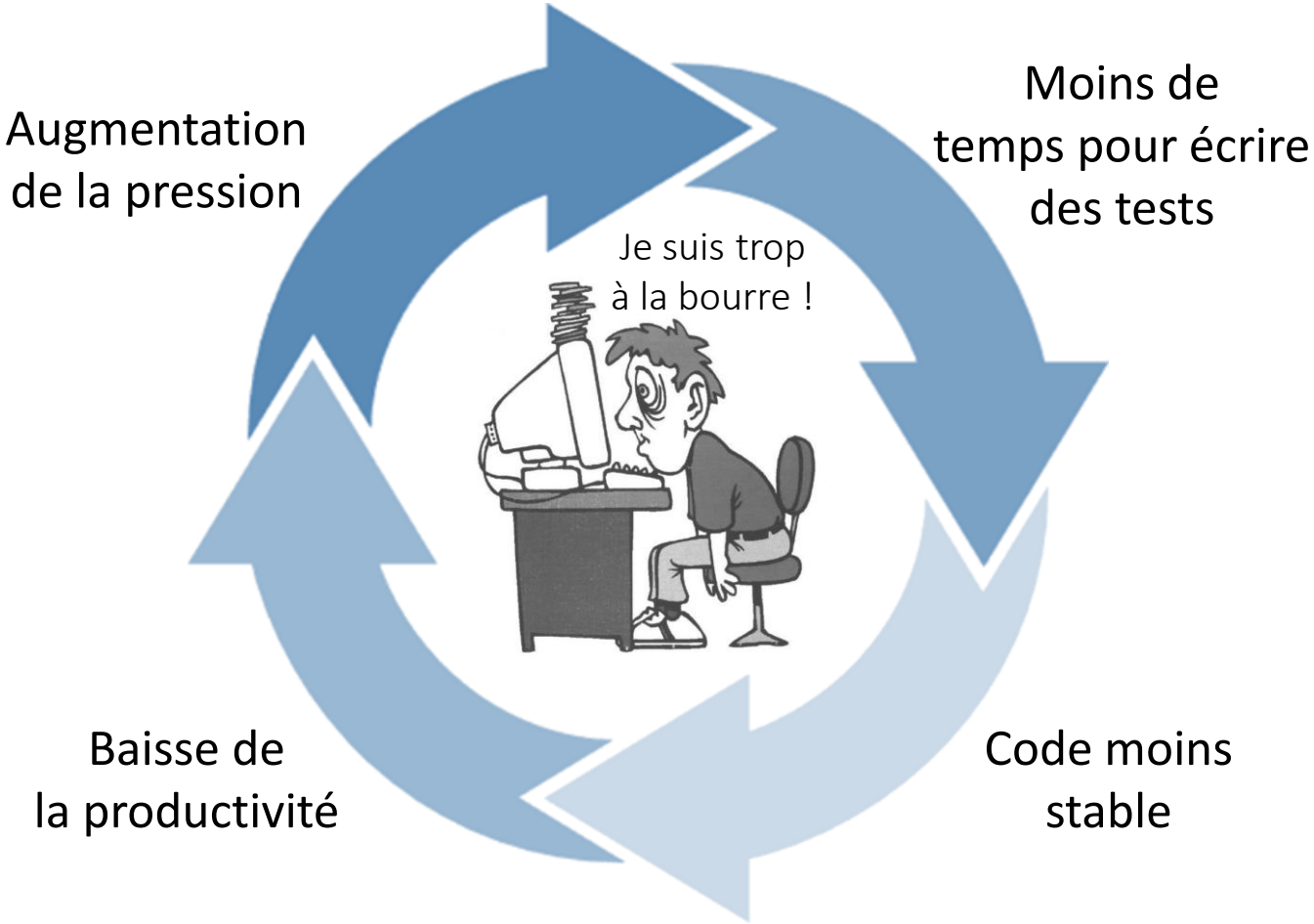
dernière modification 06/01/2025

Philippe Genoud
Philippe.Genoud@imag.fr

Le problème du test

- Tous les programmeurs savent qu'ils doivent écrire des tests, peu le font correctement..





Tests du logiciel

- différents types de test
  - test unitaire (ou test de composants)
    - procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme (appelée « unité » ou « module »).
    - en POO test du contrat de la classe (méthodes publiques et constructeurs)
  - test d'intégration
    - procédure visant à vérifier le bon fonctionnement d'un ensemble de composants formant tout ou partie d'un logiciel : dans le test d'intégration, chacun des modules indépendants du logiciel est assemblé et testé dans l'ensemble.
  - test système (ou test fonctionnel)
    - procédure visant à vérifier le bon fonctionnement du logiciel (système intégré) afin d'évaluer sa conformité aux exigences spécifiées (Spécifications fonctionnelles et techniques des besoins)
  - test de performance (benchmark)
    - utilisé pour comparer les composants logiciels à plusieurs reprises. L'objectif est de garantir que le code testé s'exécute assez rapidement, même s'il est soumis à une charge élevée.

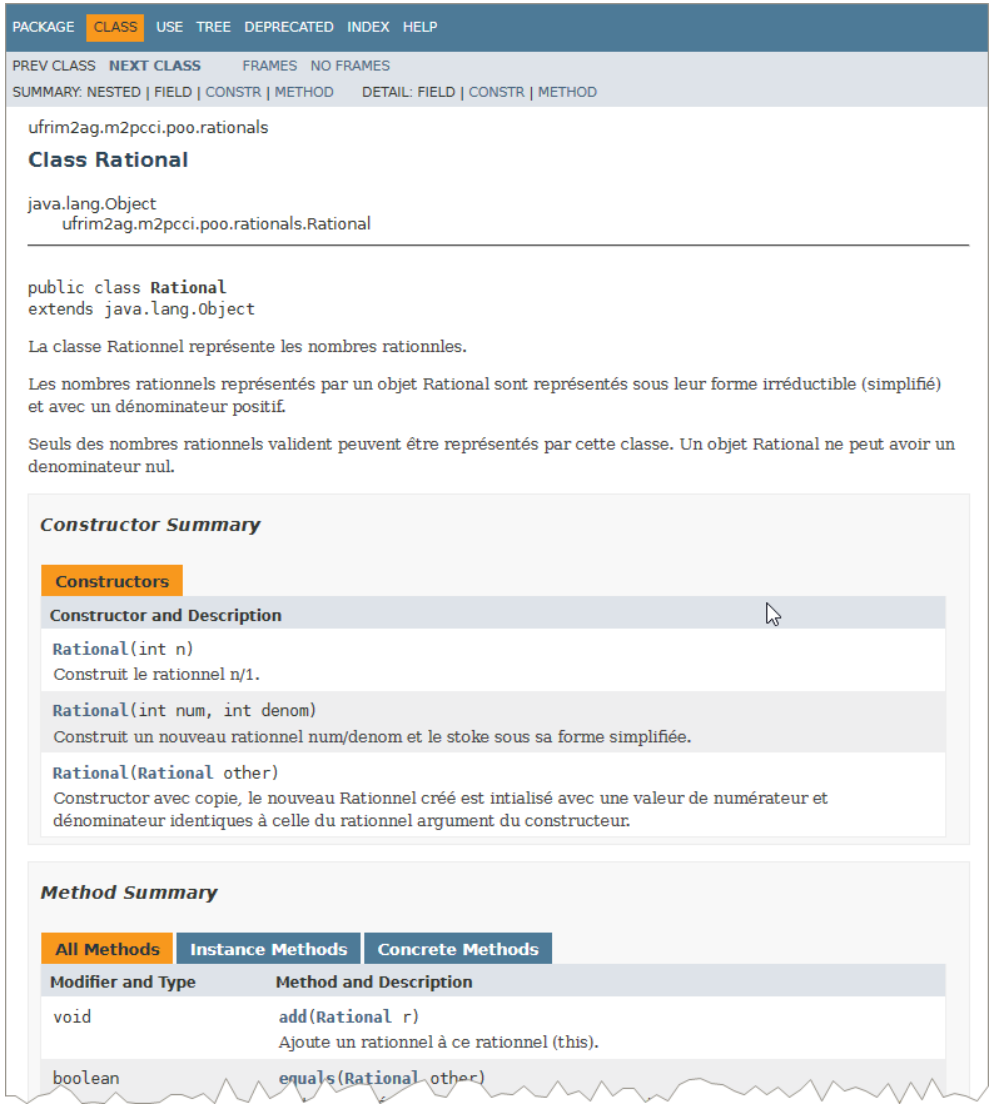
Test Unitaire d'une classe

- L'interface publique d'une classe définit "un contrat" entre celui qui fournit la classe et celui qui l'utilise.

- Ce contrat définit:
  - les services proposés par la classe
  - la manière dont ces services doivent être utilisés

- Tester une classe consiste à vérifier la validité de ce contrat
  - il confronte la réalisation de la classe à sa spécification.

→ Tests unitaires (Unit Tests)



PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

ufrim2ag.m2pcci.poo.rationals

Class Rational

java.lang.Object
ufrim2ag.m2pcci.poo.rationals.Rational

public class **Rational**
extends java.lang.Object

La classe Rationnel représente les nombres rationnles.

Les nombres rationnels représentés par un objet Rational sont représentés sous leur forme irréductible (simplifié) et avec un dénominateur positif.

Seuls des nombres rationnels valident peuvent être représentés par cette classe. Un objet Rational ne peut avoir un denominateur nul.

Constructor Summary

Constructors

Constructor and Description
`Rational(int n)` Construit le rationnel n/1.
`Rational(int num, int denom)` Construit un nouveau rationnel num/denom et le stoke sous sa forme simplifiée.
`Rational(Rational other)` Constructor avec copie, le nouveau Rationnel créé est intialisé avec une valeur de numérateur et dénominateur identiques à celle du rationnel argument du constructeur.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

Modifier and Type	Method and Description
void	`add(Rational r)` Ajoute un rationnel à ce rationnel (this).
boolean	`equals(Rational other)`

Comment tester une classe ?

- Traces (instructions **`print`**) dans les programmes

Rational.java

```
public class Rational {

    /**
     * numerateur
     */
    private int num;
    /**
     * denominateurr
     */
    private int denom;

    /**
     * Construit un nouveau rationnel num/denom et le stoke sous sa forme
     * simplifiée.
     *
```

PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

ufrim2ag.m2pcci.poo.rationals

Class Rational

java.lang.Object
ufrim2ag.m2pcci.poo.rationals.Rational

public class **Rational**
extends java.lang.Object

La classe Rationnel représente les nombres rationnles.

Les nombres rationnels représentés par un objet Rational sont représentés sous leur forme irréductible (simplifié) et avec un dénominateur positif.

Seuls des nombres rationnels valident peuvent être représentés par cette classe. Un objet Rational ne peut avoir un denominateur nul.

Constructor Summary

Constructors

Constructor and Description

Rational(int n)
Construit le rationnel n/1.

Rational(int num, int denom)
Construit un nouveau rationnel num/denom et le stoke sous sa forme simplifiée.

Rational(Rational other)
Constructor avec copie, le nouveau Rationnel créé est intialisé avec une valeur de numérateur et dénominateur identiques à celle du rationnel argument du constructeur.

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method and Description
void	**add**(Rational r) Ajoute un rationnel à ce rationnel (this).
boolean	**equals**(Rational other)

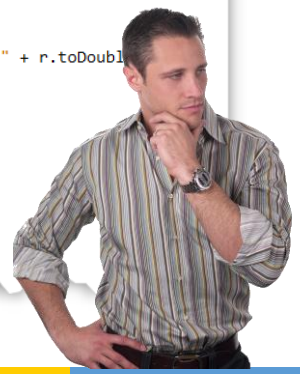
Spécification

Implémentation

Programme de test

```
public class TestRational {

    public static void main(String[] args) {
        Rational r = new Rational(6, 4);
        System.out.print("r = " + r.toString());
        System.out.println("\tsous la forme réelle, r = " + r.toDoubl

        Rational s = new Rational(2);
        System.out.println("s = " + s);

        r.add(s);
        System.out.println("r + s = " + r);

        Rational t = new Rational(34, 8);
        System.out.println("t = " + t);

        System.out.print(r + " x " + t + " = ");
```

TestRational.java



```
run:
r = 3 / 2       sous la forme réelle, r = 1.5
s = 2
r + s = 7 / 2
t = 17 / 4
2 x 17 / 4 = 17 / 2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Trace d'exécution

Vérification

Spécification

Quels sont les problèmes d'une telle approche ?

Comment tester une classe ?

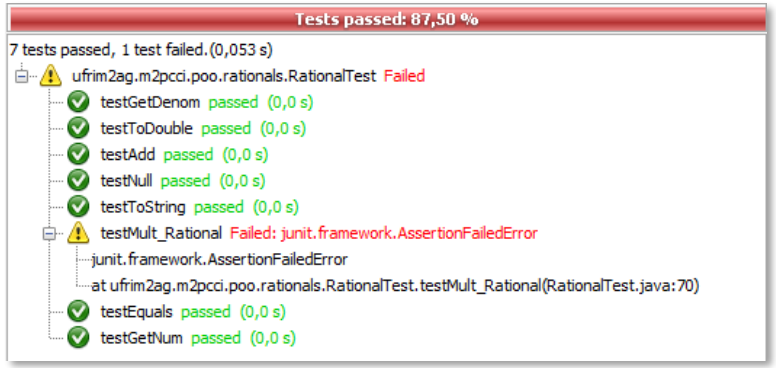
- jugement humain
  - Risques d'erreur humaine
    - si une trace contient de nombreux **`print`**, perte de lisibilité (*Scroll Blindness*)
  - Processus long et répétitif à renouveler chaque fois que la classe est modifiée et/ou qu'un bug est corrigé (test de non régression)

Inefficace et coûteux

Automatiser les tests et les confier à des programmes

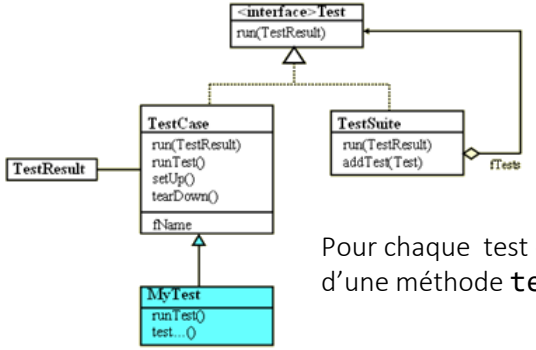
- **JUnit :** un framework open source (www.junit.org) pour le **test unitaire** de programmes Java qui permet **d'automatiser les tests.**
- **Facilite**
  - écriture des programmes de tests unitaires
  - exécution des tests unitaires
  - l'exploitation des résultats de test

- Terminologie JUnit
  - **Test unitaire (Unit test)** : test d'une classe
  - **Cas de test (Test case)** : teste les réponses d'une méthode à un ensemble particulier d'entrées
  - **Suite de tests (Test suite)** : une collection de cas de tests
  - **Testeur (Test runner)** : programme qui exécute des suites de tests et rapporte les résultats



Mise en œuvre de tests avec JUnit

- 3 versions de JUnit
  - JUnit 3.8
    - Basé sur un framework de Classes et Interfaces
    - Ecriture de tests en écrivant des classes s'intégrant au framework (par héritage)



Pour chaque test écriture d'une méthode `testXX`

  - JUnit 4 https://junit.org/junit4/
    - Basé sur des annotations (Java 5+)
    - Ces annotations permettent de marquer les classes et méthodes de test
    - Nouvelles fonctionnalités et plus de souplesse (plus de contraintes d'héritage)

JUnit

- Accroche une information à un élément (classe, méthode, attribut) Java
- Utilisée par un mécanisme tiers (compilateur java, environnement d'exécution…) … et non par la classe où elle est définie

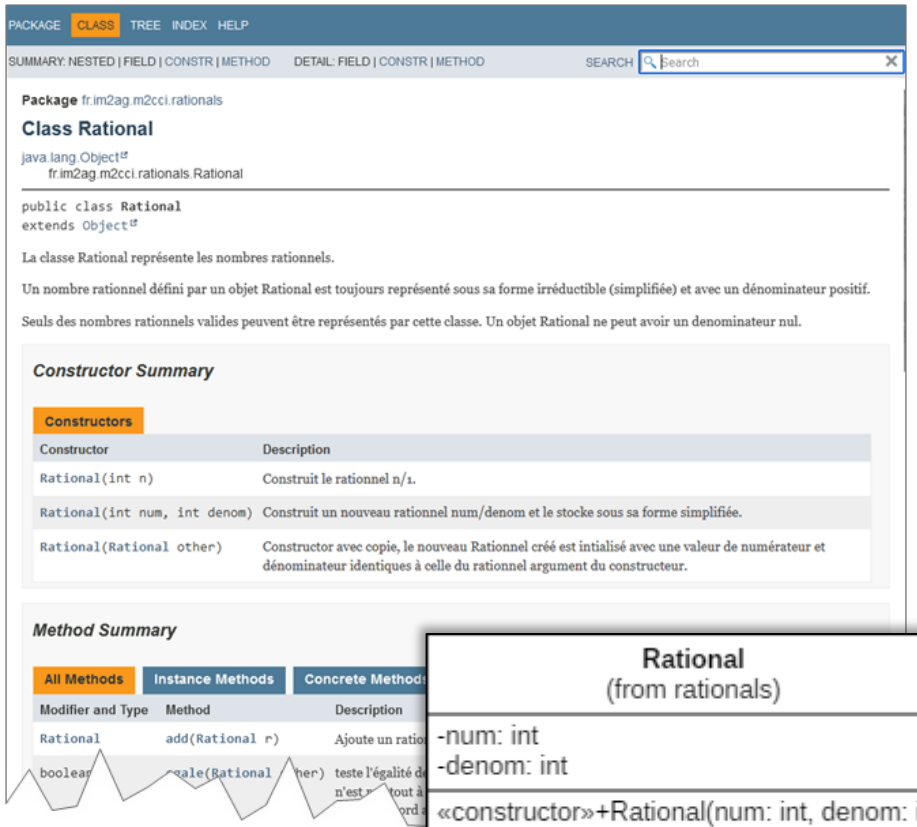
```
@Test
public void testMult() {
      ...
}
```

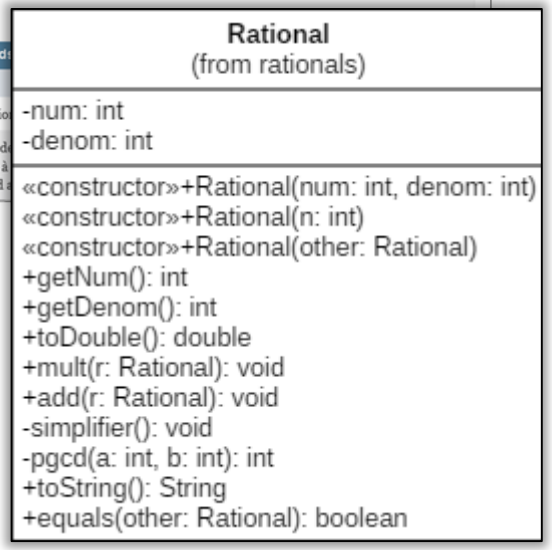
  - JUnit 5 https://junit.org/junit5/

JUnit 5 is the current generation of the JUnit testing framework, which provides a modern foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.

JUnit 5

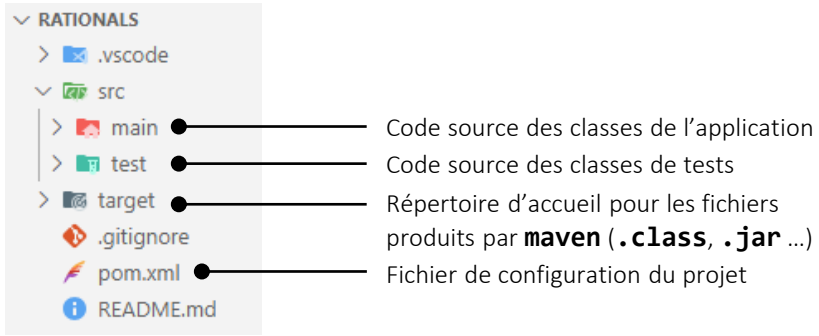
Exemple : test unitaire de la classe Rational

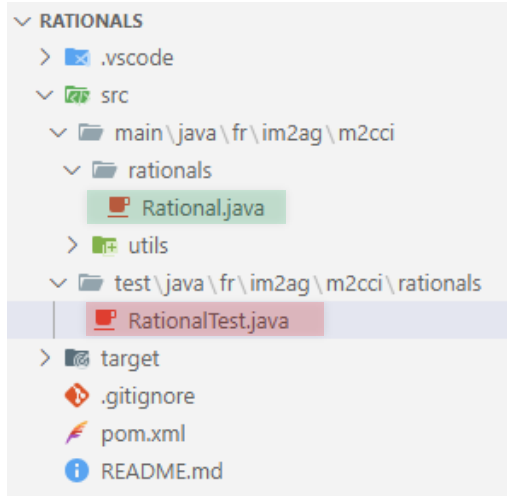


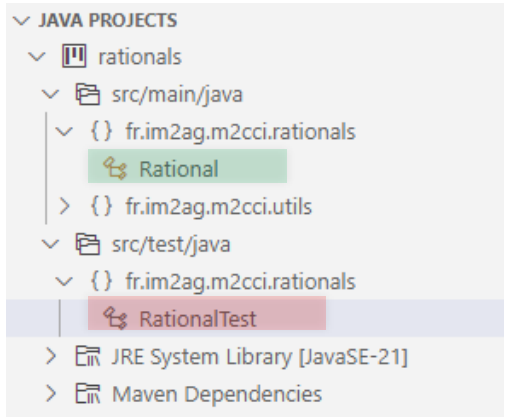


- Pour réaliser les tests unitaires d'une classe création d'une classe de test séparée
  - `Rational` la classe à tester
  - `RationalTest` la classe de test contenant les tests unitaires de `Rational`
- En général les classes de test se situent dans un dossier source différent de celui des classes à tester pour séparer le code effectif de l'application du code de test

organisation des fichiers dans un projet apache **`maven`** (outil java de construction de projets)







JUnit 5 : Ecriture d'une classe de Test

- La classe de test :

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.Assertions;
public class RationalTest {

    //…

    @Test
    public void testMult() {

        Rational r1 = new Rational(2, 3);
        Rational r2 = new Rational(3, 5);

        r1.mult(r2);

        Assertions.assertEquals(2, r1.getNum());
        Assertions.assertEquals(5, r1.getDenom());
    }
}
```

Cas de test pour la méthode `mult` de la classe `Rational`

Création des objets qui vont interagir lors du test

Code qui agit sur les objets impliqués dans le test

Vérification que le résultat obtenu correspond bien au résultat attendu.

```
static void assertEquals(int expected,
                         int actual)
```

méthode statique de [org.junit.jupiter.api.Assertions]

- vérifie que le premier paramètre (valeur attendue) est égal au deuxième paramètre (valeur calculée)
- Si la valeur calculée n'est pas égale à la valeur attendue lance une exception (en fait une `Error`) de type `AssertionFailedError`

JUnit 5 : Ecriture d'une classe de Test

- La classe de test :

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class RationalTest {

    //…

    @Test
    public void testMult() {

        Rational r1 = new Rational(2, 3);
        Rational r2 = new Rational(3, 5);

        r1.mult(r2);

        assertEquals(2, r1.getNum());
        assertEquals(5, r1.getDenom());
    }
}
```

utilisation d'un `import static` pour '*simplifier*' les appels des méthodes d'assertion

Cas de test pour la méthode `mult` de la classe `Rational`

Création des objets qui vont interagir lors du test

Code qui agit sur les objets impliqués dans le test

Vérification que le résultat obtenu correspond bien au résultat attendu.

inutile de préfixer les méthodes par le nom de la classe

```
static void assertEquals(int expected,
                         int actual)
```

méthode statique de [org.junit.jupiter.api.Assertions]

- vérifie que le premier paramètre (valeur attendue) est égal au deuxième paramètre (valeur calculée)
- Si la valeur calculée n'est pas égale à la valeur attendue lance une exception (en fait une `Error`) de type `AssertionFailedError`

JUnit 5 : Ecriture d'une classe de Test

- La classe de test :

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class RationalTest {

    //…

    @Test
    public void testMult() {

        Rational r1 = new Rational(2, 3);
        Rational r2 = new Rational(3, 5);

        r1.mult(r2);

        assertEquals(2, r1.getNum());
        assertEquals(5, r1.getDenom());
    }
}
```

utilisation d'un `import static` pour '*simplifier*' les appels des méthodes d'assertion

Cas de test pour la méthode `mult` de la classe `Rational`

Création des objets qui vont interagir lors du test

Code qui agit sur les objets impliqués dans le test

Vérification que le résultat obtenu correspond bien au résultat attendu.

inutile de préfixer les méthodes par le nom de la classe

```
static void assertEquals(int expected,
                         int actual)
```

méthode statique de [org.junit.jupiter.api.Assertions]

- vérifie que le premier paramètre (valeur attendue) est égal au deuxième paramètre (valeur calculée)
- Si la valeur calculée n'est pas égale à la valeur attendue lance une exception (en fait une `Error`) de type `AssertionFailedError`

JUnit 5 : Exécution d'une classe de Test

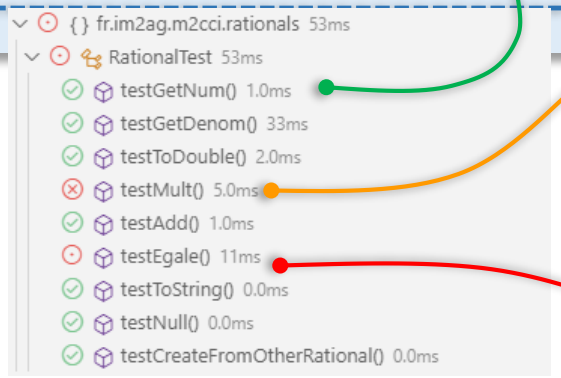
- La classe de test :

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class RationalTest {

    //…

    @Test
    public void testMult() {

        Rational r1 = new Rational(2, 3);
        Rational r2 = new Rational(3, 5);

        r1.mult(r2);

        assertEquals(2, r1.getNum());
        assertEquals(5, r1.getDenom());
    }
}
```

Cas de test pour la méthode **mult** de la classe **Rational**

Exécution automatique par l'exécuteur de tests JUnit

Rapport de tests

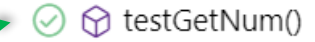


La classe de test est exécutée par le **TestRunner** de JUnit

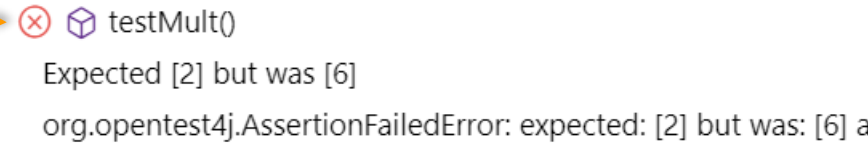
chaque cas de test est exécuté indépendamment des autres et un rapport de tests est produit.

pour chaque cas de test 3 types de résultats sont possibles :

Cas de Test réussi : toutes les assertions ont été vérifiées

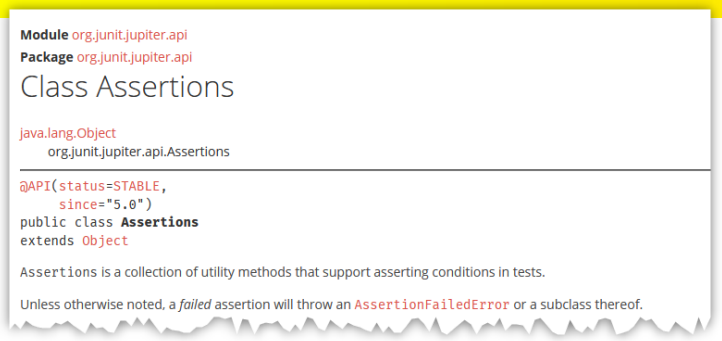


Failure: le code ne correspond pas au critères de test, une assertion n'est pas vérifiée



Erreur: une situation imprévue est intervenue dans l'exécution du test - une exception a été levée et non atrappée

JUnit 5 : Les différentes méthodes assert



Module org.junit.jupiter.api
Package org.junit.jupiter.api
Class Assertions

java.lang.Object
org.junit.jupiter.api.Assertions

```
@API(status=STABLE,
     since="5.0")
public class Assertions
extends Object
```

Assertions is a collection of utility methods that support asserting conditions in tests.

Unless otherwise noted, a *failed* assertion will throw an AssertionFailedError or a subclass thereof.

La classe **Assertions** du package **org.junit.jupiter.api** fourni de nombreuses méthodes statiques pour définir des conditions de test
https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

- `assertEquals(Object expected, Object actual)`
  `assertEquals(int expected, int actual)`
  `assertEquals(float expected, float actual)`

  Méthodes surchargées pour tous les types primitifs

  `assertEquals(float expected, float actual, float delta)`

  Lorsque `assertEquals` porte sur des valeurs `float` ou `double` possibilité de spécifier un delta qui représente un seuil de différence entre les deux valeurs

  `...`

  `assertEquals(Object expected, Object actual, String message)`
  `assertEquals(int expected, int actual, String message)`
  `assertEquals(int expected, int actual, String message)`

  Acceptent un éventuel paramètre message fournissant une description textuelle documentant l'erreur et facilitant le diagnostic des causes d'échec

  `...`

  `assertEquals(Object expected, Object actual, Supplier<String> messageSupplier)`
  `assertEquals(int expected, int actual, Supplier<String> messageSupplier)`
  `assertEquals(int expected, int actual, Supplier<String> messageSupplier)`

  Le message peut éventuellement être fourni par un `messageSupplier` (une fonction retournant une String qui ne sera évalué qu'en cas d'échec (failure))

  `...`

  - Vérifie que la valeur attendue (**expected**) est égale à la valeur effective (**actual**). Pour les objets, utilise la méthode **equals** héritée de **Object***, sinon pour les types primitifs utilise **==**.
  - Si les deux valeurs ne sont pas égales une **AssertionFailedError** est provoquée.

* ⚠ c'est la méthode de signature `public boolean equals(Object o)` qui est appelée

JUnit 5 : Les différentes méthodes assert

- `assertSame(Object expected, Object actual)`
  `assertSame(Object expected, Object actual, String message)`
  `assertSame(Object expected, Object actual, Supplier<String> messageSupplier)`
  - Vérifie que `expected` et `actual` référencent le même objet (==), sinon une `AssertionFailedError` est provoquée.

- `assertNotSame(Object expected, Object actual)`
  `assertNotSame(Object expected, Object actual, String message)`
  `assertNotSame(Object expected, Supplier<String> messageSupplier)`
  - Vérifie que `expected` et `actual` ne référencent pas le même objet (!=), sinon une `AssertionFailedError` est provoquée.

- `assertNull(Object object)`
  `assertNull(Object object, String message)`
  `assertNull(Object object, Supplier<String> messageSupplier)`
  - Vérifie que object == null, sinon une `AssertionFailedError` est provoquée.

- `assertNotNull(Object object)`
  `assertNotNull(Object object , String message)`
  `assertNotNull(Object object , Supplier<String> messageSupplier)`
  - Vérifie que la référence `object` n'est pas la valeur `null`, sinon une `AssertionFailedError` est provoquée.

JUnit 5 : Les différentes méthodes assert

- `static void assertTrue(boolean test)`
  `static void assertTrue(boolean test, String message)`
  `static void assertTrue(boolean test, Supplier<String> messageSupplier)`
  - Vérifie que `test == true` sinon une `AssertionFailedError` est provoquée.

- `static void assertFalse(boolean test)`
  `static void assertFalse(boolean test, String message)`
  `static void assertFalse(boolean test, Supplier<String> messageSupplier)`
  - Vérifie que `test == false`, sinon une `AssertionFailedError` est provoquée.

- `assertArrayEquals` compare le contenu d'un tableau réel à un tableau attendu.
- `assertLinesMatch` compare deux listes de Strings.
- `assertTimeout` vérifie that que la function fournie se termine avant le timeout spécifié
- …

- `fail()`
  `fail(String message)`
  `fail(Supplier<String> messageSupplier)`
  - Provoque l'échec du test et lance une `AssertionFailedError`
  - Utile lorsque les autres méthodes `assert` ne correspondent pas exactement à vos besoins ou pour tester que certaines exceptions sont bien lancées.

JUnit 5: tester les exceptions

"Verifying that code completes normally is only part of programming. Making sure the code behaves as expected in exceptional situations is part of the craft of programming too. » JUnit Cookbook Kent Beck, Erich Gamma

- Comment vérifier que des exceptions sont lancées comme prévu ?

```
@Test
public void testDenomNull() {

       try {
         new Rational(3,0);
         fail("IllegalArgumentException non lancée");
       }
       catch (IllegalArgumentException e) {
          // OK exception lancée
       }
}
```

à la JUnit 3.8

```
try {
  // appel d'une méthode devant lancer une Exception
  ...
  // si l'exception n'a pas eu lieu on force le
  //test  échouer
  fail("Did not throw an ExpectedException");
}
catch (ExpectedException e) {
}
```

JUnit 5

Vérifie que l'exception est bien levée et renvoie l'objet exception

La classe de l'exception

Expression lambda contenant le code à exécuter provoquant l'exception

```
@Test
 public void testDenomNull() {
    IllegalArgumentException except = assertThrows(IllegalArgumentException.class, () -> new Rational(10, 0) );
    assertEquals("dénominateur nul interdit", except.getMessage());
}
```

JUnit5 : factoriser du code entre les méthodes de test

```java
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestRational {
    // ...

    @Test
    public testMult() {

        Rational r1 = new Rational(2, 3);
        Rational r2 = new Rational(7, 5);
        r1.mult(r2);
        assertEquals(14,r1.getNum());
        assertEquals(15,r1.getDenom());
    }


    @Test
      public void testAdd() {

        Rational r1 = new Rational(2, 3);
        Rational r2 = new Rational(7, 5);
        r1.add(r2);
        assertTrue(31 == r1.getNum());
        assertTrue(15 == r1.getDenom());
    }

}
```

```java
private Rational r1;
private Rational r2;
```

Code exécuté avant chaque méthode de test

```java
@BeforeEach
public void setUp() {
    r1 = new Rational(2, 3);
    r2 = new Rational(7, 5);
}
```

Code exécuté après chaque méthode de test

```java
@AfterEach
public void tearDown() {
    ...
    ...
}
```

JUnit5 : factoriser du code entre les méthodes de test

Code exécuté **une seule fois** avant de commencer l'exécution des méthodes de test de la classe

Code exécuté **une seule fois** après avoir exécuté toutes les méthodes de test de la classe

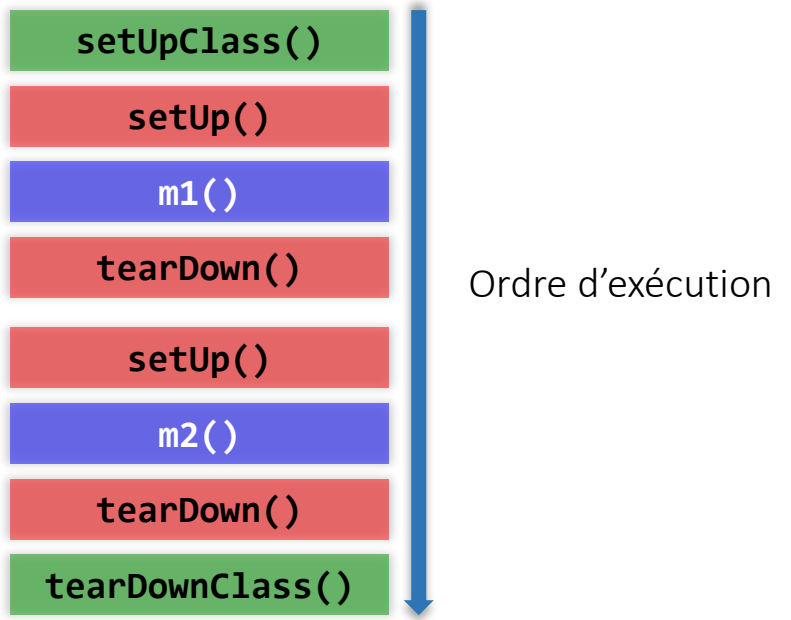
Code exécuté **avant** chaque méthode de test de la classe

Code exécuté **après** chaque méthode de test de la classe

méthode de test

méthode de test

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;
public class TestCounter {

    @BeforeAll
    public static void setUpClass() {
        ...
    }

    @AfterAll
    public static void tearDownClass() {
        ...
    }

    @BeforeEach
    public void setUp() {
      ...
    }

    @AfterEach
    public void tearDown() {
      ...
    }

    @Test
    public void m1() {
       ...
    }

    @Test
        public void m2() {
       ...
    }
}
```



On ne peut préjuger d'un ordre d'exécution des méthodes de test (`@Test`)

JUnit5 : Exécution d'une classe de test

- en mode texte sur la console

```
P:\ENSEIGNEMENT\ServeursWEB\teaching\PL2AI_2019_2020\tds\PL2\sujets\tp06_introJUnit\TP06_RationalsJunit
λ mvn clean test
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building TP06_RationalsJUnit 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------

[INFO] Surefire report directory: P:\ENSEIGNEMENT\ServeursWEB\teaching\PL2AI_2019_2020\tds\PL2\sujets\tp06_introJUnit\TP06_RationalsJunit\target\surefire-reports

-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running fr.im2ag.cci.rationals.RationalTest
getNum
add
getDenom
toDouble
egale
toString
Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.039 sec

Results :

Tests run: 8, Failures: 0, Errors: 0, Skipped: 0

[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 3.871 s
[INFO] Finished at: 2019-12-19T12:43:13+01:00
[INFO] Final Memory: 16M/369M
[INFO] ------------------------------------------------------------------------

P:\ENSEIGNEMENT\ServeursWEB\teaching\PL2AI_2019_2020\tds\PL2\sujets\tp06_introJUnit\TP06_RationalsJunit
λ
```

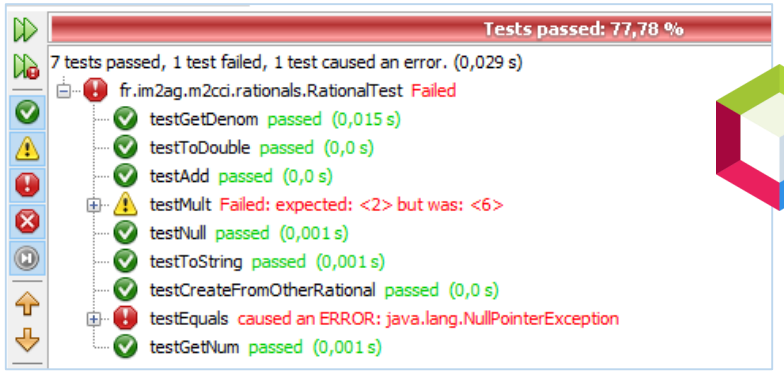
Commande maven, qui efface tout ce qui a été construit, recompile et lance les tests

nombre de méthodes de test ayant échoué à cause d'une exception levée et non traitée

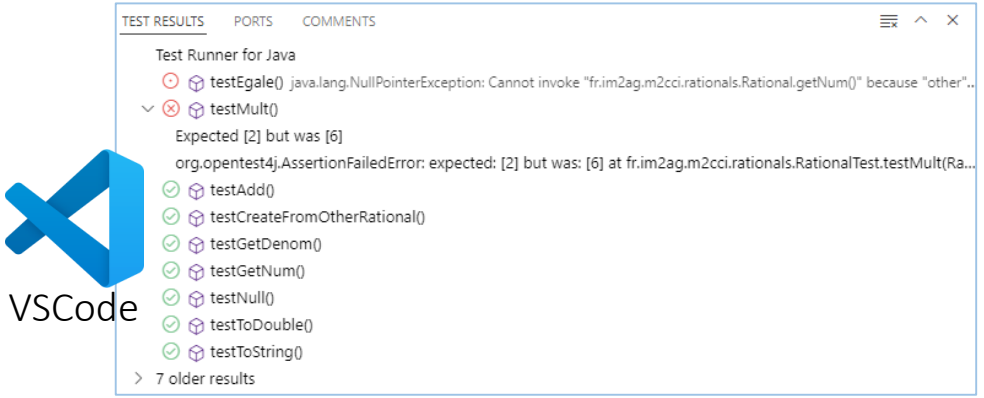
nombre de méthodes de test exécutées

nombre de méthodes de test ayant échoué (assertion non vérifiée ,méthode `fail`)

De nombreux IDE intègrent JUnit : IntelliJ IDEA, NetBeans, Eclipse, VSCode…



NetBeans



VSCode

JUnit 5 : Exécuter un ensemble de tests

- dans JUnit 5 possibilité de définir des suites de tests (TestSuite) pour exécuter ensemble un ensemble de tests en agrégeant plusieurs classes de test.
- JUnit 5 fourni les annotations suivantes :
  - `@SelectPackages` – pour spécifier les noms des packages pour la suite de tests
  - `@SelectClasses` – pour spécifier les classes pour la suite de tests. elles peuvent être situées dans différents packages.

- un exemple utilisant les deux annotations pour sélectionner deux classes de test et un package:

TestSuiteDemo.java

```
package fr.im2ag.m2cci.rational;

import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.IncludeTags;
import org.junit.platform.suite.api.SelectClasses;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;


@RunWith(JUnitPlatform.class)
@SelectClasses( { RationalTest.class, ComplexTest.class } )
@SelectPackages("fr.im2ag.m2cci.geometry")
public class TestSuiteDemo {
}
```

classes dont tous les tests sont à exécuter

ces classes peuvent elles-mêmes être des suites de de tests

package dont toutes les classes de test sont à exécuter

Couverture de code

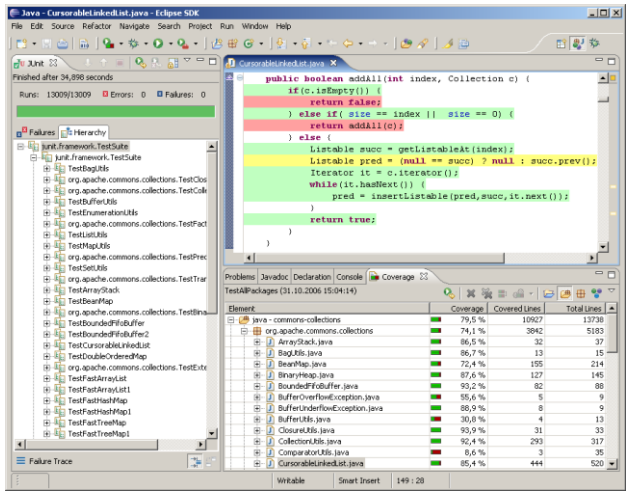
- comment mesurer la qualité des test effectués ?
- un indicateur de la qualité des tests effectués peut être la **couverture de code** (*code coverage*)
  - une mesure utilisée en génie logiciel pour décrire le taux de code source testé d'un programme.
  - nombreuses méthodes pour mesurer la couverture de code. Les principales sont :
    - Couverture des fonctions (Function Coverage)
      Chaque fonction dans le programme a-t-elle été appelée ?
    - Couverture des instructions (Statement Coverage)
      Chaque ligne du code a-t-elle été exécutée et vérifiée ?
    - Couverture des points de tests (Condition Coverage)
      Chaque point d'évaluation (tel que le test d'une variable) a-t-il été exécuté et vérifié ? (Le point de test teste-t-il ce qu'il faut ?)
    - Couverture des chemins d'exécution (Path Coverage)
      Chaque parcours possible (par exemple les 2 cas vrai et faux d'un if) a-t-il été exécuté et vérifié ?

Couverture de code

- Très souvent en complément d'un framework de tests unitaires on utilise des outils de couverture de code
  - JaCoCo Java Code Coverage Library http://www.eclemma.org/jacoco/
    - Intégration dans les IDE, ou avec maven

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.jacoco.examples		58%		64%	24	53	97	193	19	38	6	12
org.jacoco.core		97%		93%	107	1,378	115	3,311	20	710	2	136
org.jacoco.agent.rt		77%		84%	31	121	62	310	21	74	7	20
jacoco-maven-plugin		90%		80%	37	186	46	412	8	111	0	19
org.jacoco.cli		97%		100%	4	109	10	275	4	74	0	20
org.jacoco.report		99%		99%	4	572	2	1,345	1	371	0	64
org.jacoco.ant		98%		99%	4	163	8	429	3	111	0	19
org.jacoco.agent		86%		75%	2	10	3	27	0	6	0	1
Total	1,355 of 27,205	95%	146 of 2,129	93%	213	2,592	343	6,302	76	1,495	15	291

EclEmma for Eclipse http://www.eclemma.org/

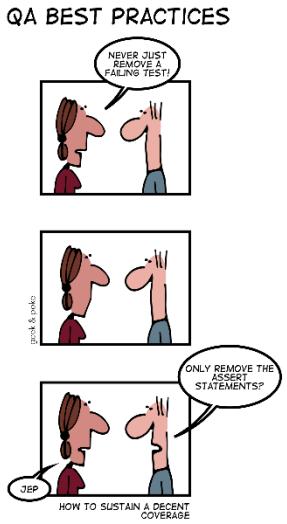


- *d'autres outils Java*
  - Clover https://www.atlassian.com/software/clover
  - OpenClover https://openclover.org/
  - Cobertura http://cobertura.github.io/cobertura/
  - ....

Dashboard | Application code | Test code | Test results | Top risks | Quick wins | Coverage tree map

Project	Tests	Fail	Error	Time	% Pass
Clover database Sat Sep 26 2015	978	9	0	491.12	99.1%

Packages	Tests	Fail	Error	Time(secs)	% Pass
com.atlassian.clover.cfg.instr	7	4	0	0.057	42.9%
com.atlassian.clover.ant	21	2	0	0.793	90.5%
com.atlassian.clover.ant.tasks	104	3	0	84.478	97.1%
com.atlassian.clover.instr.groovy	170	0	0	291.291	100%
com.atlassian.clover.test	1	0	0	1.51	100%
com.atlassian.clover.ci	48	0	0	0.059	100%
com.atlassian.clover.ant.types	3	0	0	0.004	100%
com.atlassian.clover.ant.groovy	11	0	0	47.006	100%
com_atlassian_clover	11	0	0	0.038	100%
com.atlassian.clover.util.trie	16	0	0	0.181	100%



JUnit Testing tips (JUnit primer)

> *"Any program feature without an automated test simply doesn't exist."*
>
> Extreme Programming Explained, Kent Beck

- Code a little, test a little, code a little, test a little . . .
- Run your tests as often as possible, at least as often as you run the compiler ☺
- Begin by writing tests for the areas of the code that you're the most worried about . . .write tests that have the highest possible return on your testing investment
- When you need to add new functionality to the system, write the tests first
  - TDD Test Driven Development : écrire les tests avant les implémentations des classes
- If you find yourself debugging using `System.out.println()`, write a test case instead
- When a bug is reported, write a test case to expose the bug
- Don't deliver code that doesn't pass all the tests

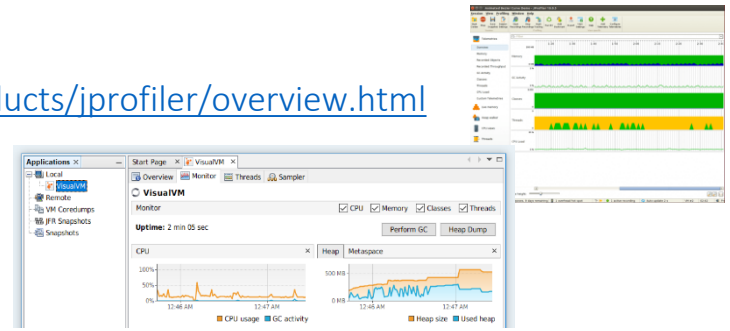
www.extremeprogramming.org

Tests et Intégration Continue

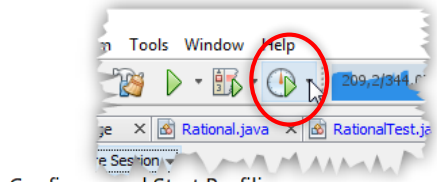
- Intégration continue (CI)
  - pratique de développement logiciel où les développeurs intègrent fréquemment leur code dans un dépôt central. Chaque intégration est ensuite vérifiée par une compilation automatique et des tests automatisés.
  - points clés :
    - **Automatisation des tests** : Chaque modification du code déclenche une série de tests pour s'assurer que les nouvelles modifications n'introduisent pas de bugs.
    - **Détection rapide des erreurs** : En intégrant fréquemment, les erreurs sont détectées plus tôt, ce qui facilite leur correction.
    - **Amélioration de la collaboration** : Les développeurs travaillent sur une base de code commune, ce qui réduit les conflits de fusion et améliore la collaboration.
    - **Déploiement plus rapide** : Les équipes peuvent déployer des fonctionnalités plus rapidement et plus fréquemment.
- L'objectif principal de l'intégration continue est d'améliorer la qualité du logiciel et de réduire le temps nécessaire pour livrer des mises à jour

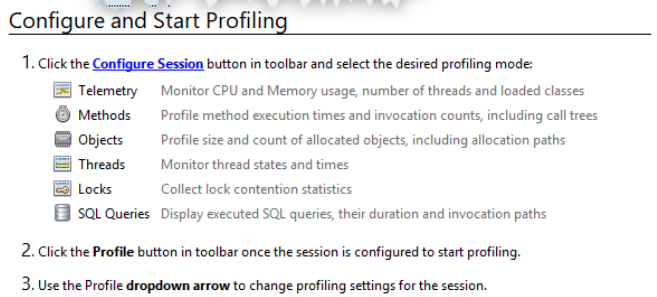
JUnit et les autres

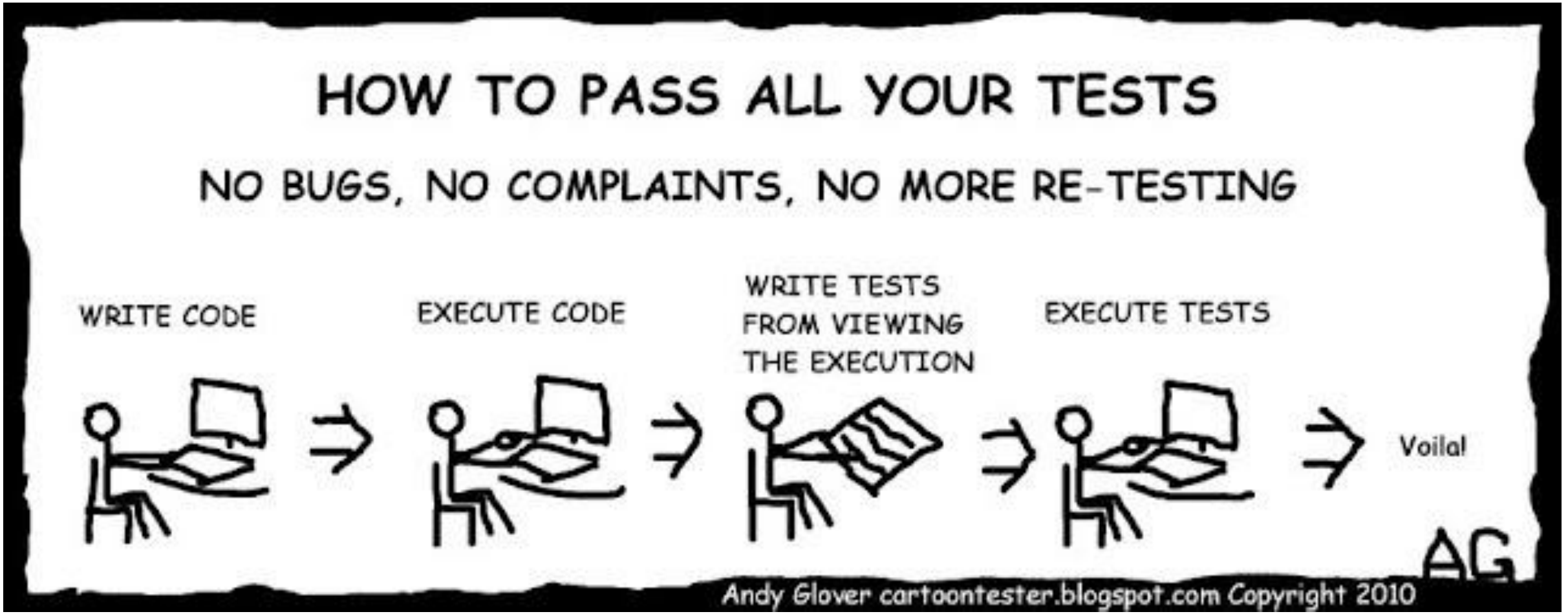
- xUnit famille de frameworks pour le test unitaire automatisés.
  - Disponibles pour de nombreux langages et environements:
    - JUnit. (Java) http://www.junit.org.
    - TestNG (java) http://testng.org/doc/index.html
      http://kaczanowscy.pl/tomek/sites/default/files/testng_vs_junit.txt.slidy_.html#%281%29
    - cppUnit (C++). https://sourceforge.net/projects/cppunit/
    - nUnit (.NET) https://nunit.org/
    - ...
- D'autres outils
  - couverture de code (JaCoco , clover ...)
  - SeleniumHQ (test applications Web)
    - https://www.selenium.dev/
    - https://github.com/SeleniumHQ/seleniumhq.github.io
  - outils de profiling
    - https://www.baeldung.com/java-profilers
    - JProfiler https://www.ej-technologies.com/products/jprofiler/overview.html
    - VisualVM https://visualvm.github.io/index.html
    - ...



outil de profiling intégré à NetBens

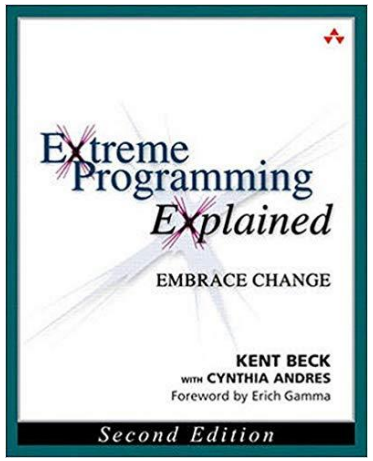





HOW TO PASS ALL YOUR TESTS
NO BUGS, NO COMPLAINTS, NO MORE RE-TESTING
WRITE CODE
EXECUTE CODE
WRITE TESTS FROM VIEWING THE EXECUTION
EXECUTE TESTS
Voila!
AG
Andy Glover cartoontester.blogspot.com Copyright 2010

Références

- Le site JUnit : www.junit.org
- JUnit 5 Tutorial: Writing Assertions With JUnit 5 Assertion API
  https://www.petrikainulainen.net/programming/testing/junit-5-tutorial-writing-assertions-with-junit-5-api/
- Maven – JaCoCo code coverage example, mkyong , published November 15, 2018
  https://www.mkyong.com/maven/maven-jacoco-code-coverage-example/
- Junit 5 – Développons en Java J.M. DOUDOUX
  https://www.jmdoudoux.fr/java/dej/chap-junit5.htm
- Le site www.extremeprogramming.org
- eXtreme Programming Explained: Embrace Change & Test Driven Design
  - by Cynthia Andres, Kent Beck
  - Publisher: Addison-Wesley Professional
  - Release Date: November 2004
  - ISBN: 0321278658



JUnit 3/4: tester les exceptions

"Verifying that code completes normally is only part of programming. Making sure the code behaves as expected in exceptional situations is part of the craft of programming too. » JUnit Cookbook Kent Beck, Erich Gamma

- Comment vérifier que des exceptions sont lancées comme prévu ?

```
@Test
public void testDenomNull() {

       try {
          new Rational(3,0);
          fail("IllegalArgumentException non lancée");
       }
       catch (IllegalArgumentException e) {
          // OK exception lancée
       }
}
```

à la JUnit 3.8

```
try {
  // appel d'une méthode devant lancer une Exception
  ...
  // si l'exception n'a pas eu lieu on force le
  //test  échouer
  fail("Did not throw an ExpectedException");
}
catch (ExpectedException e) { }
```

Avec JUnit4 paramètre **expected** de l'annotation **@Test**

```
@Test (expected = IllegalArgumentException.class)
 public void testDenomNull() {
   new Rational(3,0);
}
```

Le test échoue si l'exécution de la méthode dépasse le temps fixé par le paramètre timeout

```
@Test(timeout=10)
public void uneMéthode() {
  ...
}
```