IUT Nantes
Pôle Sciences et technologie

Nantes Université

R2.03 - Qualité de développement 1
Automatisation des tests

CM4 Implémentation de tests unitaires Kotlin avec JUnit

Kotlin et les tests

- Rappel : Kotlin est compilé en bytecode et exécuté par la JVM comme Java
- Kotlin peut continuer à utiliser le framework de test (par défaut) de Java : JUnit
- Kotlin propose aussi son propre framework de test : Kotest

JUnit

- Origine
  - Extreme programming (test-first development)
  - framework de test écrit en Java par E. Gamma et K. Beck
  - open source : www.junit.org

- Généralisation des concepts (xUnit) :
  - Architecture introduite en 1994 avec SUnit (Smalltalk)
  - https://en.wikipedia.org/wiki/XUnit

- Objectifs
  - test d'applications en Java
  - faciliter la création des tests
  - tests de non régression

Codage JUnit5 (1/4)

- Organisation du code des tests
  - Méthode de test :
    - Chaque méthode de test annotée avec @Test implémente un seul cas de test
    - Chacune contient : description, initialisation, appel avec donnée de test, oracle d'un cas de test (sauf cas des tests paramétriques cf. plus loin)
  - Classe de Test : (TestCase dans le vocabulaire de Junit)
    - Contient les méthodes de test des différents cas de test
    - setUp() et tearDown() annoté avec @BeforeEach @AfterEach
      - Appelée avant et après (resp.) chaque méthode de test
      - Peut factoriser l'initialisation de plusieurs méthodes de test
    - une classe de test définie une suite de cas de test

Codage (2/4)

- Codage d'un « TestCase » Junit dans un programme Kotlin:
  - déclaration de la classe:

```
import org.junit.jupiter.api.Test
import org.junit.jupiter.api.BeforeEach
import org.junit.jupiter.api.AfterEach

internal class OperationsTest {

    val op = Operations() // instance de la classe sous test

    @BeforeEach
    fun setUp() {...}

    @AfterEach
    fun tearDown() {...}

    @Test
    fun testAdditionner1() {      ...    }
}
```

Codage (3/4)

- la méthode setUp avec @BeforeEach:
  - //appelée avant chaque cas de test

```
@BeforeEach
fun setUp() {
    //réinitialisation d'attribut d'objet
    //rappel de constructeur, etc.
}
```

- la méthode tearDown avec @AfterEach:
  - //appelée après chaque cas de test

```
@ AfterEach
fun tearDown() {
        //libération de variable
        //remise à jour de BDD, etc.
}
```

Codage (4/4)

- les méthodes de test
  - chaque méthode implémente un et un seul cas de test
- caractéristiques:
  - nom préfixé par « test » par convention
  - Annotation @Test
  - contient obligatoirement un oracle programmé
    - Par défaut, il faut donc au moins une assertion
      - Obligatoire (sauf pour le test des levées d'exceptions)

```
@Test
fun testAdditionner1() {
   val op = Operations() // initialisation (qui aurait pu être anticipée)
   val res = op.additionner(0, 0) // lancement du test :
                        appel de la méthode sous test avec la donnée de test
   assertEquals(0, res, "0+0=0") // oracle: vérification active du résultat attendu
}
```

Les assertions

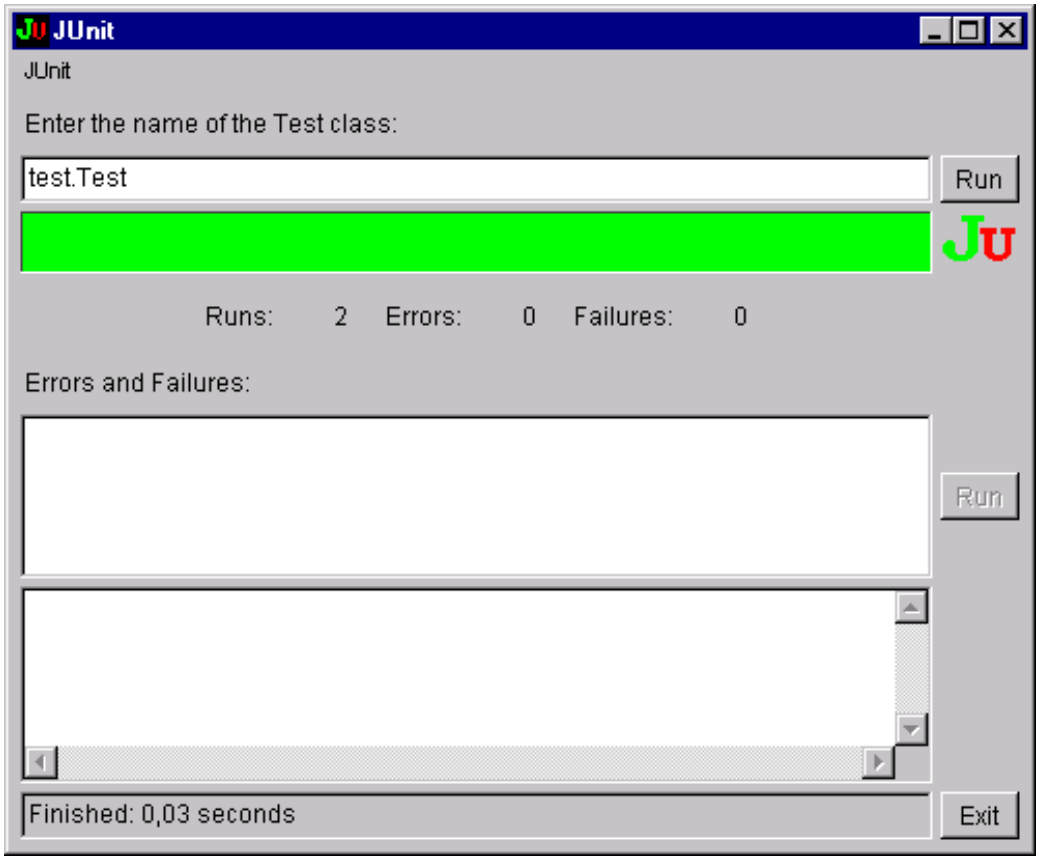
- assertEquals(Object expected, Object actual, String msg)
- assertSame(Object exp, Object act, String msg)
- assertEquals(Object exp [], Object act [], String msg)
- assertEquals(float exp, float act , float delta, String msg)
- assertTrue(boolean b, String msg)
- assertFalse(boolean b, String msg)
- assertNull(Object o, String msg)
- assertNotNull(Object o, String msg)
- fail(String msg)

- + des variantes (sans msg), etc.

Verdict des tests JUnit

- pass
- failure
  - Une assertion n'est pas satisfaite
  - Révèle un bug
- error
  - Le test n'a pas pu terminer
    - Test erroné
    - Ou programme erroné
- Dans tous les cas, méfiance :
  - Les tests peuvent eux-mêmes être mal écrits
    - Mettant l'équipe projet en confiance à tort
    - Il faut d'abord bien comprendre les tests

TestRunner


JUnit
JUnit
Enter the name of the Test class:
test.Test
Run
Runs: 2 Errors: 0 Failures: 0
Errors and Failures:
Run
Finished: 0,03 seconds
Exit

Intégration dans IntelliJ