IUT Nantes
Pôle Sciences et technologie

Nantes Université

Test paramétrique

Tests paramétriques

- constatation : beaucoup de répétition de code quand on teste de grande séries de valeurs
- définition de patrons de code pour les classes de test, pour éviter la duplication du code

- Junit 5 intègre nativement la possibilité de faire des tests paramétriques paramétrés de multiple manière. (contrairement à Junit 4 qui possède dans sa librairie une fonctionnalité pour implémenter des tests paramétriques, mais souffrant de lacunes, nous lui préférons JUnitParams : http://pragmatists.github.io/JUnitParams/)

- CsvSource
- Permettant de passer plusieurs paramètres par test mais limité à des types convertissables depuis un String :

```
import org.junit.jupiter.params.ParameterizedTest
import org.junit.jupiter.params.provider.CsvSource

internal class OperationsTest {

        val op = Operations()

    @ParameterizedTest
    @CsvSource(
       "1, 2, 3",
       "2, 3, 5"
    )
    fun testAdditionBinaire(dt1: Int, dt2: Int, oracle: Int){
       assertEquals(oracle, op.additionnerBinaire(dt1, dt2))
    }
}
```

Permettant de passer plusieurs paramètres de tous types :

```
import org.junit.jupiter.params.ParameterizedTest
import org.junit.jupiter.params.provider.Arguments
import org.junit.jupiter.params.provider.MethodSource
import java.util.stream.Stream

internal class OperationsTest {
    val op = Operations()

  @ParameterizedTest
  @MethodSource("intTabProviderAdditioner")
  fun testAdditionBinaire2(dt1: Int, dt2: Int, oracle: Int, message: String){
      assertEquals(oracle, op.additionnerBinaire(dt1, dt2), message)
  }

companion object {//nécessaire en kotlin pour utiliser des méthodes static (répandues en Java)
    @JvmStatic
    fun intTabProviderAdditioner(): Stream<Arguments?>? {
        return Stream.of(
            Arguments.of(1, 2, 3,"1+2=3"),
            Arguments.of(2, 3, 5, "2+3=5")
        )
    }
}
```

Bénéfices de framework de test unitaire comme JUnit

- Simple à comprendre : méthode, classe et suites de test.
- Simple à utiliser: @Test, @BeforeEach, @AfterEach, etc.
- Structuré : cas de test, suite de tests.
- Exécution de programme simple et reproductible permettant le débogage
- Permet de sauvegarder les cas de test :
  - indispensable pour la non régression
    - quand une classe évolue, on ré-exécute les cas de test.
- Plusieurs extensions : tests de BD et IHM, rapports, etc.