

# FRAMEWORK JUNIT



**JUnit**  
Testing Framework

**Bureau E204**

# PLAN DU COURS

- Introduction
- Tests Unitaires
- Utilisation de JUNIT
- Place à la Pratique

# INTRODUCTION

- Il existe différents niveaux de test :
  - Test unitaire
  - Test d'intégration
  - Test de charge
  - Test fonctionnel
  - Test sécurité
  - ....

# TEST UNITAIRE : DÉFINITION

- Un test unitaire est une procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel. Il s'agit d'un code.
- En POO, on teste au niveau des classes
- Pour chaque classe (MyClass), on a une classe de test (MyClassTest).

# TEST UNITAIRE : QUELQUES RÈGLES

- Doit être isolé : il doit être indépendant
- N'est pas un test de bout en bout : il agit que sur une portion de code
- Doit être déterministe : le résultat doit être le même pour les mêmes entrées
- Est le plus petit et simple possible

# TEST UNITAIRE : QUELQUES RÈGLES

- Ne teste pas d'enchaînement d'actions
- Etre lancé le plus souvent possible : intégration continue
- Etre lancé le plus tôt possible : détection des bug plus rapide
- Couvrir le plus de code possible
- Etre lancé a chaque modification

# TEST UNITAIRE : AVANTAGE ET INTÉRÊT

- Garantie la non régression
- Détection de bug plus facile
- Aide à isoler les fonctions
- Aide à voir l'avancement d'un projet (TDD)

\* Le **test-driven development (TDD)** ou en français développement piloté par les tests est une technique de développement de logiciel qui préconise d'écrire les tests unitaires avant d'écrire le code source d'un logiciel.

# TEST UNITAIRE : OUTIL DE TEST

PHP	JS	SQL	JAVA
PHPUnit SimpleTest	JSUnit	SQLUnit	JUnit



# TEST UNITAIRE : CAS A TESTER

- Lors de l'utilisation de test unitaire on se doit de tester différents cas.
  - Cas en succès : fonctionnement normal
  - Cas d'erreur : test sur la gestion d'erreur

# TEST UNITAIRE : LES RÉSULTATS

- Un test unitaire peut renvoyer 3 résultats différents :
  - Success : test réussi
  - Error : erreur inattendue à l'exécution
  - Failure : au moins une assertion est fausse

# TEST UNITAIRE : MOCK

- Quelques fois un test a besoin d'un composant donné pour s'exécuter.
- Par exemple pour tester une fonctionnalité, nous avons besoin du retour d'un Web Service, qui n'a toujours pas été développé.
- Il est alors utile d'utiliser des bouchons (MOCK) pour isoler le test.
- De plus un bouchon permet de tester tout les cas (valeur correcte, erroné etc.)

# UTILISATION DE JUNIT

- Il n'y a pas de limite au nombre de tests au sein de notre classe de test.
- On écrit au moins un test par méthode de la classe testée.
- Pour désigner une méthode comme un test, il suffit d'utiliser l'annotation **@Test** (à partir de JUnit4).

# UTILISATION DE JUNIT

- Au sein des tests, on utilise des **assertions** pour valider ou non un test. Quelques assertions indispensables :

Assertion	Action
<code>assertEquals()</code>	Vérifie l'égalité entre deux entités
<code>assertNotEquals()</code>	Vérifie l'inégalité entre deux entités
<code>assertFalse()</code>	Vérifie que la valeur fourni en paramètre est fausse
<code>assertTrue()</code>	Vérifie que la valeur fourni en paramètre est vrai
<code>assertNull()</code>	Vérifie que la valeur fourni en paramètre est l'objet NULL
<code>assertNotNull()</code>	Vérifie que la valeur fourni en paramètre n'est pas l'objet NULL

# UTILISATION DE JUNIT

```
package tn.esprit.esponline;

import org.apache.log4j.Logger;

public class Calcul {

    private static final Logger logger = Logger.getLogger(Calcul.class);

    public int calculerSom (int a, int b) {
        int res = 0;
        try {
            logger.info("In calculerSom(" + a + ", " + b + ")");
            res = a+ b;
            logger.info("Out calculerSom() : " + res);
        } catch (Exception e) {logger.error("Erreur : " + e);}
        return res;
    }
}
```

# UTILISATION DE JUNIT

(Suite)

```
public int calculerDiff (int a, int b)
{
    int res = 0;
    try {
        logger.info("In calculerDiff(" + a + ", " + b + ")");
        res = a - b;
        logger.info("Out calculerDiff() : " + res);
    }
    catch (Exception e) { logger.error("Erreur : " + e);
    }
    return res;
}
```

# UTILISATION DE JUNIT (Annotations)

```
package tn.esprit.esponline;

import org.junit.Assert.assertEquals;
import org.junit.Test;

public class Calcul1Test {

    Calcul calcul = new Calcul();

    @Test
    public void testCalculerSom() {
        assertEquals(25, calcul.calculerSom(10, 15));
    }

    @Test
    public void testCalculerDiff() {
        assertEquals(30, calcul.calculerDiff(40, 10));
    }

}
```



# JUNIT : PLACE A LA PRATIQUE

---

**Développement d'un Projet qui implémente JUNIT**

**Ajout de JUNIT dans un Projet déjà existant**

**QUIZ**