JUnit

JUnit est un framework de tests unitaires dédié à Java. La présente fiche vaut pour **JUnit 4** dans l'environnement **NetBeans**.

Création d'une classe de tests

Sélectionner la classe à tester puis, dans le menu **Tools** choisir **Create / Update Tests**	
Dans la boite de dialogue qui s'ouvre, choisir impérativement JUnit**4** et valider.	

Il est aussi possible de créer une **Suite de tests** qui fabriquera la classe de test de chaque classe existante dans un package. Une suite de test permettra alors de lancer l'ensemble des tests en une seule demande.

Création d'une méthode test

Une méthode de test est faite pour tester une méthode de la classe testée. On aura donc un lien direct entre la méthode testée et sa méthode de test. Pour concrétiser cette réalité, une convention consiste à appliquer le même nom aux deux méthodes puis à préfixer ou postfixer celui de la méthode de test par le phonème “test”.

On pourra aussi avoir plusieurs méthodes de test pour une seule méthode à tester. Dans ce cas, on enrichira le nom de chaque méthode de test avec un descriptif aussi clair et concis que possible.

Une méthode de test est une procédure (type retour : **void**) qui n'accepte **aucun paramètre**.

```
@Test
public void testToHTML_CasGeneral()
{
    Object[] unTableau = {"A","B","C"};
    int n = 3;
    String expected =
"<table><tr><td>A</td></tr><tr><td>B</td></tr><tr><td>C</td></tr></table>";
```

```
    String actual = HTMLUtil.toHtml (unTableau, n);
    Assert.assertEquals (expected, actual);
}
```

Assertions

Les assertions sont l'outil qui permet de raisonner sous la forme de “Cas de test” : **une situation en entrée doit produire une situation en sortie**. Les assertions permettent de vérifier chaque cas en vérifiant que la situation effective en sortie est conforme à celle attendue.

Assertion	Descriptif
assertTrue et **assertFalse**	Vérifient qu'une donnée résultat est bien, respectivement, Vraie ou Fausse
assertNull	Vérifie qu'une donnée résultat est Null
assertEquals et **assertSame**	Vérifient qu'une donnée résultat est bien égale à une donnée attendue. La seconde est plutôt à usage des tableaux et objets.
assertThrows	Vérifie qu'une exception est bien produite.
fail	Fait échouer le test sans condition. Utilisé notamment comme indicateur qu'un test n'est pas encore totalement implémenté.

Activation / Désactivation d'une méthode de test

Pour activer une méthode de test, il faut la faire précéder par l'annotation **@Test**.
Pour désactiver une méthode de test, il faut la faire précéder par l'annotation **@Ignore**.
L'absence d'annotation @Test produit le même effet que @Ignore mais n'est pas conseillé car non-explicite.

Traitements pré-test et post-test

Dans certains cas, il sera nécessaire de réaliser des traitements avant ou après chaque test.
Les méthodes **setUp** et **tearDown** associées respectivement aux annotations **@Before** et **@After** peuvent être utilisées pour réaliser un traitement qui s'exécute avant ou après chaque test.

Dans le même esprit, les méthodes **setUpClass** et **tearDownClass** associées respectivement aux annotations **@BeforeClass** et **@AfterClass** peuvent être utilisées dans le but d'exécuter un traitement avant ou après les tests mais, dans ce cas, une seule fois pour toute la batterie des tests.

```
@BeforeClass
public static void setUpClass() {
}

@AfterClass
public static void tearDownClass()
{
}

@Before
public void setUp() {
}

@After
public void tearDown() {
}
```

Exécution d'une classe de tests

L'exécution d'une classe de test se fait en cliquant-droit sur cette classe (ou la classe testée) et en choisissant **Test File**.

Toutes les méthodes de test annotées @Test sont exécutées mais l'ordre de leur exécution n'est pas garanti.

Les résultats sont présentés en fin d'exécution dans l'onglet **Tests results**.