Laboratoire: Tests unitaires

# Objectif(s)

* Créer des tests unitaires:
  + Méthodes
  + Exception
  + Déterminer des valeurs de tests

# Créer la branche de la séance

## Création du Repository et de la branche de la séance

1. Dans Gitkraken, clonez votre repository GitHub **A21\_3W6\_Labos\_NOM\_PRENOM**
2. Cliquez sur la branche de base: **MAIN**
3. Créez la branche de la séance: **S06\_TestsUnitaires**

**NUnit**

# Créer un projet de tests NUNIT

## Créer le projet pour Les Tests NUNIT

1. Créez un projet de type Tests NUnit Net 5.0 Core **App\_TestsUnitaires\_Tests\_NU**
2. À partir du projet **App\_TestsUnitaires\_Tests\_NU,** ajoutez une référence au projet: **Appl\_TestsUnitaires**

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** INIT Création projet Tests NUnit
   * **Description**: Création du projet de tests NUnit
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Tester FizzBuzz 1 méthode par scénario

## Identifiez les scénarios à tester

1. Examinez la méthode à tester: **GetOutput**
2. Identifiez les scénarios à tester pour couvrir toutes les classes d'équivalence et des DONNÉES (valeurs) de tests:
   * Divisible par 3 et 5: 15
   * Divisible par 3: 3
   * Divisible par 5: 5
   * Non divisible par 3 ou 5 : 7

## Tester avec divisible par 3 et 5

1. Renommez la classe de tests générée par défaut **E1\_FizzBuzz\_Tests**
2. Supprimez la méthode Setup (pas utile pour l'instant)

Suggestion: Affichez la classe **E1\_FizzBuzz** à côté de la classe de tests (clic droit sur une des classe, Nouveau groupe de documents vertical)

1. Renommez la méthode Test1: GetOutput\_Duv3And5\_ReturnFizzBuzz
2. Placez en commentaires les trois sections A:

public void GetOutput\_Div3And5\_ReturnFizzBuzz()

{

// Arrange

// Act

// Assert

Assert.Pass();

}

1. Dans Arrange, générez une variable Valeur = 15
2. Dans Act, exécutez la méthode de FizzBuzz GetOutput en passant Valeur en paramètre : Vous devez ajouter une référence de classe au namespace Appl\_TestsUnitaires (Using)
3. Créez une variable résultat pour mettre le résultat final du Act: ici une seule étape donc on peut faire 1 seule ligne

var Result = E1\_FizzBuzz.GetOutput(Valeur);

1. Dans Assert, remplacez **Assert.Pass()** (sinon le test réussi tout le temps) par la comparaison du Result et du résultat attendu.

// Assert

// par défaut: sensible aux majuscules/minuscules

Assert.That(Result, Is.EqualTo("FizzBuzz"));

1. Exécutez le test

## Tester avec divisible par 3 seulement

1. Créez la méthode de test: GetOutput\_Div3\_ReturnFizz
2. Placez en commentaires les trois sections A:

public void GetOutput\_Div3\_ReturnFizz ()

{

// Arrange

// Act

// Assert

Assert.Pass();

}

1. Complétez les sections du test:
2. Exécutez le test
3. Faites une erreur volontaire dans le résultat attendu: fizz en minuscule
4. Exécutez le test
5. Ajoutez IgnoreCase

// Insensible aux majuscules/minuscules

Assert.That(Result, Is.EqualTo("fizz").IgnoreCase);

1. Exécutez le test

## Tester avec divisible par 5 seulement

1. Créez la méthode de test: GetOutput\_Div5\_ReturnBuzz
2. Placez en commentaires les trois sections A:

public void GetOutput\_Div5\_ReturnFizz ()

{

// Arrange

// Act

// Assert

Assert.Pass();

}

1. Complétez les sections du test:
2. Exécutez le test

## Tester avec NON divisible par 3 ou 5

Suggestion: Affichez la classe **E1\_FizzBuzz** à côté de la classe de tests (clic droit sur une des classe, Nouveau groupe de documents vertical)

1. Créez la méthode de test: GetOutput\_NonDiv3OrDiv5\_ReturnParametre
2. Placez en commentaires les trois sections A:

public void GetOutput\_Div3\_ReturnFizz ()

{

// Arrange

// Act

// Assert

Assert.Pass();

}

1. Complétez les sections du test: Attention au type de données de retour: 7 versus "7"
2. Exécutez le test

## Commentaires et validation (Commit) des changements dans le code

1. Dans GitKraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Tests BuzzFizz
   * **Description**: Tests des scénarios BuzzFizz 1 méthode par scénario
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Tester FizzBuzz 1 méthode plusieurs scénarios

## Tester FizzBuzz

1. Créez une nouvelle classe public de tests : **E1\_FizzBuzz\_Tests\_v2**
2. Ajoutez la décoration [TestFixture] à la classe et la référence au framework NUnit
3. Ajoutez la référence au projet Appl\_TestsUnitaires
4. Ajoutez la méthode de tests GetOutPut\_Div3OrAnd5\_ReturnFizzBuzz, avec la décoration (annotation) [Test]
5. Placez en commentaires les trois sections A:
6. Ajoutez les quatre TestCase et passez-les en paramètres dans la méthode

[Test]

[TestCase(15, "FizzBuzz")]

[TestCase(3, "Fizz")]

[TestCase(5, "Buzz")]

[TestCase(7, "7")]

public void GetOutPut\_Div3OrAnd5\_ReturnFizzBuzz(int response, string expectedResult)

{

// Arrange

1. Le Arrange est vide: remplacé par les testCase, dans ce cas-ci.
2. Complétez les étapes du test

[Test]

[TestCase(15, "FizzBuzz")]

[TestCase(3, "Fizz")]

[TestCase(5, "Buzz")]

[TestCase(7, "7")]

public void GetOutPut\_Div3OrAnd5\_ReturnFizzBuzz(int response, string expectedResult)

{

// Arrange

// Act

var result = FizzBuzz.GetOutput(response);

// Assert

Assert.That(result, Is.EqualTo(expectedResult));

}

1. Exécutez le tests

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Tests BuzzFizz
   * **Description**: Tests des scénarios BuzzFizz une méthode plusieurs scénarios
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Tester EvenNumbers

## Identifiez les scénarios à tester

1. Examinez la méthode à tester: **GetEvenNumbers**

La méthode retourne la liste des nombres pairs entre 0 et la valeur donnée inclusivement (noter que 0 est un nombre pair).

1. Identifiez les scénarios à tester pour couvrir toutes les classes d'équivalence et des DONNÉES (valeurs) de tests:
   * Valeur paire supérieure à 0 : 6 = » la liste retournée devrait être 0, 2, 4, 6)
   * Valeur impaire supérieure à 0 : 7
   * Valeur paire égale à 0 : 0
   * Valeur paire inférieure à 0 : -6

Ne faites que le premier scénario.

## Tester Evennumbers

1. Créez une nouvelle classe public de **E2\_EvenNombers\_Tests**
2. Ajoutez la décoration [TestFixture] à la classe et la référence au framework NUnit
3. Ajoutez la méthode de tests GetEvenNumbers\_LimitIsGreaterThanZero\_ReturnNumberUpToLimit, avec la décoration (annotation) [Test]
4. Placez en commentaires les trois sections A:
5. Ajoutez une méthode de **SetUp,** avec la décoration SetUp
   * Instancier la classe dans une varable \_evenNumbers

[SetUp]

public void SetUp()

{

\_evenNumbers = new E2\_EvenNumbers();

}

1. Dans la méthode de test, ajoutez dans ACT l’exécution de la méthode avec la valeur choisie (6)
2. Dans ASSERT, valider le résultat obtenu :
   * Que la liste n’est pas vide
   * Que la liste compte 4 éléments
   * Que la liste contient en quatre lignes : 0, 2, 4, 6
   * Que la liste contient 0, 2, 4, 6 en une seule ligne
   * Que le résultat est ordonné
   * Que les valeurs retournées sont unique

// Assert

// valide seulement si contenu. trop general

Assert.That(result, Is.Not.Empty);

// plus specifique mais ne valide pas le contenu

Assert.That(result.Count(), Is.EqualTo(4));

// Plus specifique et valide le contenu methode longue

Assert.That(result, Does.Contain(0));

Assert.That(result, Does.Contain(2));

Assert.That(result, Does.Contain(4));

Assert.That(result, Does.Contain(6));

// Plus specifique et valide le contenu methode courte

Assert.That(result, Is.EquivalentTo(new[] { 2, 0, 4, 6 }));

//// Methode qui valide l ordre

Assert.That(result, Is.Ordered);

Assert.That(result, Is.Unique);

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Tests EnvenNumbers
   * **Description**: Tests de plusieurs aspects EnvenNumbers un scénario
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

# Tester AttackPointsCalculator

## Identifiez les scénarios à tester

1. Examinez la méthode à tester: **CalculateAttackPoints**

La méthode calcule le pointage en fonction du nombre d’attaque porté. Si le nombre d’attaque est inférieur au minimum (20) aucun point est accordé. Si le nombre d’attaque est entre le minimum et le maximum, un point est accordé pour chaque cinq attaques portées de plus que le minimum. Si le nombre d’attaque porté est inférieur à 0 ou supérieur au maximum, une exception OutOfRange est levée.

1. Identifiez les scénarios à tester pour couvrir toutes les classes d'équivalence et des DONNÉES (valeurs) de tests:
   * Inférieur à la limite minimum et inférieur au maximum :
     + Nombre d’attaque 0 : 0
     + Nombre d’attaque 6 : 0
     + Nombre d’attaque 19 : 0
   * Égal à la limite minimum et inférieur au maximum :
     + Nombre d’attaque 20 : 0
   * Supérieur à la limite minimum et inférieur au maximum :
     + Nombre d’attaque 21 : 6
     + Nombre d’attaque 25 : 1
     + Nombre d’attaque 30 : 2
     + Nombre d’attaque 85 : 13
   * Inférieur au minimum
     + Nombre d’attaque -1 : ExceptionOutOfRange
   * Supérieur au maximum
     + Nombre d’attaque 205 : ExceptionOutOfRange

## Tester AttackPointsCalculator : calculer le pointage

1. Créez une nouvelle classe public de **E3\_AttackPoints\_Tests**
2. Ajoutez la décoration [TestFixture] à la classe et la référence au framework NUnit
3. Ajoutez la méthode de tests AttackPointsCalculator\_ReturnPoints, avec la décoration (annotation) [Test]
4. Placez en commentaires les trois sections A:
5. Ajoutez une méthode de **SetUp,** avec la décoration SetUp
   * Instancier la classe dans une varable \_zombieAttack

[SetUp]

public void SetUp()

{

\_zombieAttack = new E3\_AttackPointsCalculator();

}

1. Listez les scénarios qui effectuent le calcul. Avec TestCase en identifiant le nombre d’attaque portée et le résultat attendu.

[Test]

[TestCase(0, 0)]

[TestCase(6, 0)]

[TestCase(19, 0)]

[TestCase(20, 0)]

[TestCase(21, 0)]

[TestCase(25, 1)]

[TestCase(30, 2)]

[TestCase(85, 13)]

public void AttackPointsCalculator\_ReturnPoints(int nbrAttack, int expectedResult)

{

1. Dans la méthode de test, ajoutez dans ACT l’exécution de la méthode avec les valeurs de scénarios

//Act

var result = \_zombieAttack.CalculateAttackPoints(nbrAttack);

1. Dans ASSERT, valider le résultat obtenu :

// Assert

Assert.That(result, Is.EqualTo(expectedResult));

## Tester AttackPointsCalculator lever une exception

1. Créez une nouvelle classe public de **E3\_AttackPoints\_Tests**
2. Ajoutez la décoration [TestFixture] à la classe et la référence au framework NUnit
3. Ajoutez la méthode de tests AttackPointsCalculator\_ThrowArgumentOutOfRangeException, avec la décoration (annotation) [Test]
4. Placez en commentaires les trois sections A:
5. Listez les scénarios qui effectuent le calcul. Avec TestCase

[Test]

[TestCase(-1)]

[TestCase(205)]

public void AttackPointsCalculator\_ThrowArgumentOutOfRangeException(int nbrAttack)

1. Effectuez le ACT et ASSERT dans la même ligne de code qui lève une exception

// Act et Assert

Assert.That(() => \_zombieAttack.CalculateAttackPoints(nbrAttack), Throws.Exception.TypeOf<ArgumentOutOfRangeException>());

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Tests AttackPointsCalculator
   * **Description**: Tests AttackPointsCalculator, tous les scénarios et une exception
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 

**XUnit**

# Créer un projet de tests XUNIT

## Créer le projet pour Les Tests XUNIT

1. Créez un projet de type Tests NUnit Net 5.0 Core **App\_TestsUnitaires\_Tests\_XU**
2. À partir du projet **App\_TestsUnitaires\_Tests\_XU,** ajoutez une référence au projet: **Appl\_TestsUnitaires**

## Reproduire certains tests en xunit

1. Créez une classe de test XUnit **E1\_FizzBuzz \_Tests\_V2**
2. Créez une classe de test XUnit **E3\_AttackPoints\_Tests**

## Commentaires et validation (Commit) des changements dans le code

1. Dans Gitkraken, assurez-vous d'être dans le bon dossier/repo et sur la bonne branche
2. Cliquez sur **View changes**
3. Validez les modifications: cliquez sur **Stage all changes**
4. Commentez en respectant les standards proposés (vous serez noté là-dessus):
   * **Summary** FCT Tests en XUnit
   * **Description**: Tests en XUnit
5. Cliquez sur **Commit**
6. "Poussez" **Push** les modifications sur le repo en ligne **Remote**: cliquez sur 