ULB

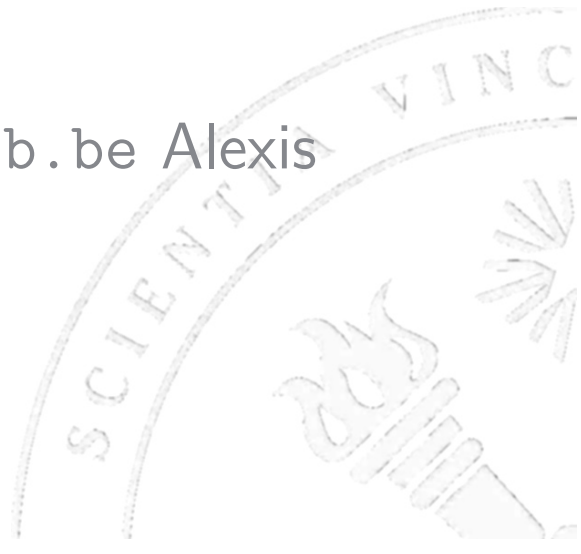
UNIVERSITÉ LIBRE DE BRUXELLES

INFO-F-307 - Séance 2
Planification Itérative & Test Driven Development

Anthony Cnudde - anthony.cnudde@ulb.be
Abel Laval - abel.laval@ulb.be
Yannick Molinghen - yannick.molinghen@ulb.be Alexis Reynouard - alexis.reynouard@ulb.be

Université Libre de Bruxelles
A.A. 2021-2022

15 février 2022

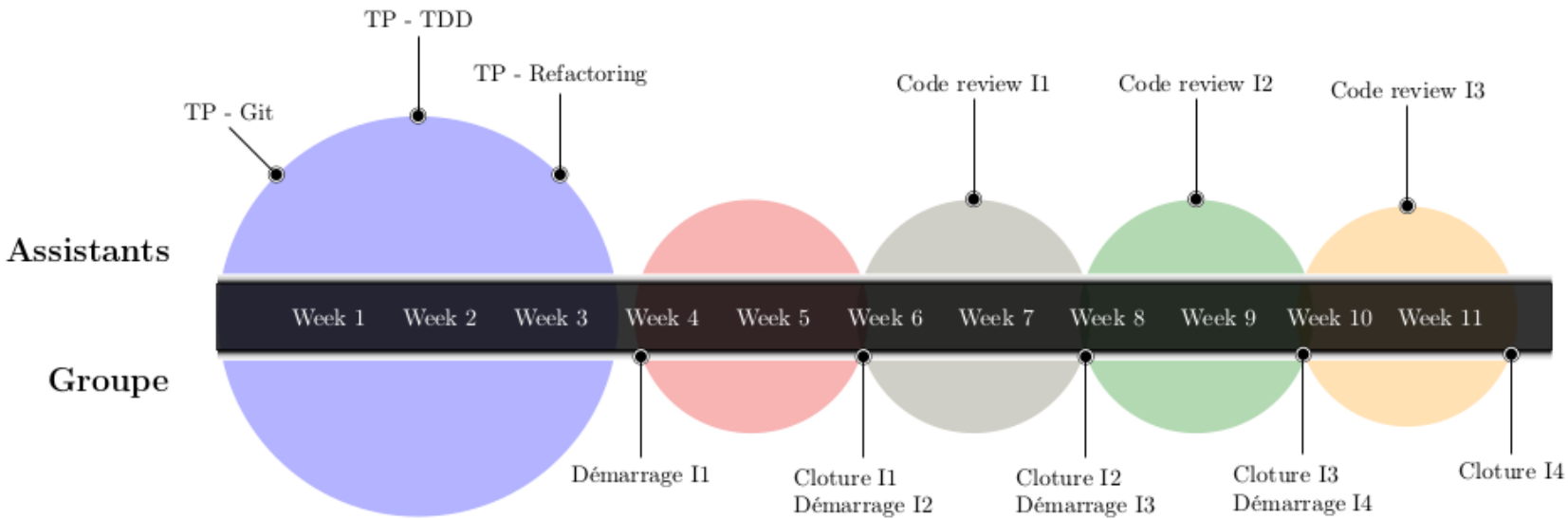


ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Planification itérative

Les livraisons/itérations



- Dans ce cours, un librable ≡ une itération

Les histoires

Histoire : 3 Groupe :

Titre : Importation et exportation des fichiers

Description : L'utilisateur peut importer plusieurs fichiers à la fois ou un répertoire ou une archive compressée (*.tar.gz*) à partir de son ordinateur. Les fichiers déjà chargés par l'utilisateur via le répertoire ne devraient pas être chargés une seconde fois.

Priorité Client : **2**
Risque Développeurs :
Introduit dans l'iteration :
État :
Points :
Notes :

- Points ≡ Heures de travail **par binôme**
- Risque → **1, 2, 3 ≡ Fort, Moyen, Faible**

Les histoires

Histoire : 3 Groupe :

Titre : Importation et exportation des fichiers

Description : L'utilisateur peut importer plusieurs fichiers à la fois ou un répertoire ou une archive compressée (*.tar.gz*) à partir de son ordinateur. Les fichiers déjà chargés par l'utilisateur via le répertoire ne devraient pas être chargés une seconde fois.

Priorité Client : **2**
Risque Développeurs : **A completer avant la réunion**
Introduit dans l'iteration :
État :
Points : **A completer avant la réunion**
Notes :

- Points ≡ Heures de travail **par binôme**
- Risque → **1, 2, 3 ≡ Fort, Moyen, Faible**

Les histoires

Histoire : 3 Groupe :

Titre : Importation et exportation des fichiers

Description : L'utilisateur peut importer plusieurs fichiers à la fois ou un répertoire ou une archive compressée (*.tar.gz*) à partir de son ordinateur. Les fichiers déjà chargés par l'utilisateur via le répertoire ne devraient pas être chargés une seconde fois.

Priorité Client : **2**
Risque Développeurs : **3**
Introduit dans l'iteration : **A completer pendant la réunion**
État : **A completer pendant la réunion**
Points : **20**
Notes : **A completer pendant la réunion**

- Points ≡ Heures de travail **par binôme**
- Risque → **1, 2, 3 ≡ Fort, Moyen, Faible**

Du point de vue développeur

while $!isProjectFinished$ **do**
    Exploration()
    Engagement()
    Suivi()
end while

Méthodologie

Du point de vue développeur

while $!isProjectFinished$ **do**
    Exploration()
    Engagement()
    Suivi()
end while

Méthodologie

$\Rightarrow$

while $!isProjectFinished$ **do**
    for all histoires do
        Division en tâches
        Estimation points
    end for
    for histoires choisies do
        Répartition des tâches
        Suivi
    end for
end while

Dans la pratique

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Wrap-up

- **Avant la réunion**
  - Estimation des points et des risques **pour toutes les histoires**
  - Division en tâches

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Wrap-up

- **Avant la réunion**
  - Estimation des points et des risques **pour toutes les histoires**
  - Division en tâches
- **Pendant la réunion**
  - Discussion
  - Suivi de la discussion
  - Choix des histoires **par le client**

ULB
UNIVERSITÉ LIBRE DE BRUXELLES

Wrap-up

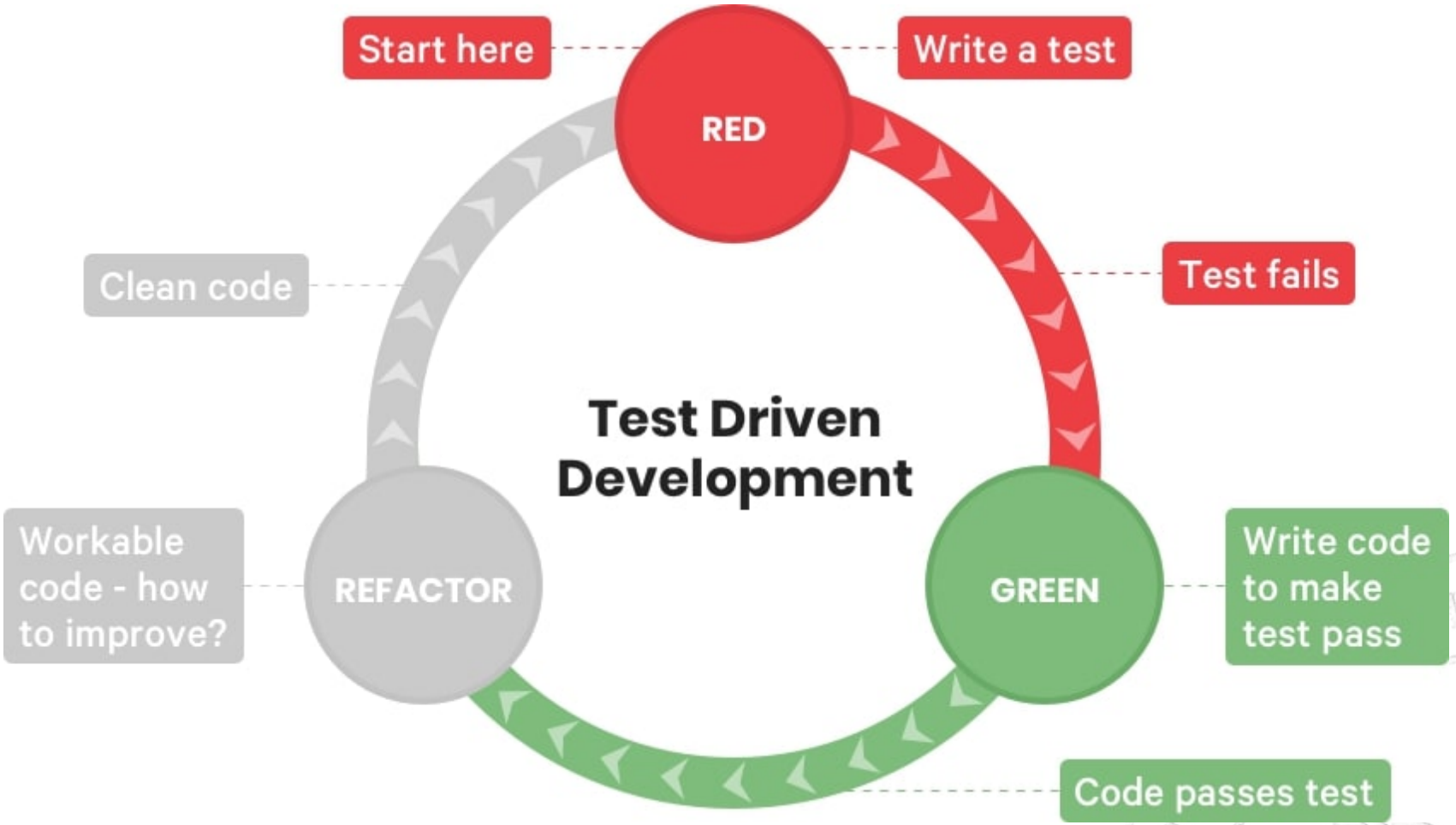
- **Avant la réunion**
  - Estimation des points et des risques **pour toutes les histoires**
  - Division en tâches
- **Pendant la réunion**
  - Discussion
  - Suivi de la discussion
  - Choix des histoires **par le client**
- **Après la réunion**
  - Assignation des tâches
  - Développement
  - Suivi des tâches

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

TDD - Méthodologie

Le processus



Comment tester avant de coder ?

Liste des tests ⇒ Processus itératif

1. Identifier tâches d'implémentation
2. Tâche atomique (i.e. totalement évidente) ?
   - Oui ⇒ Stop
   - Non ⇒ Décomposition
3. Si, dans la liste, il existe des tâches non atomiques, répéter point 2. pour chaque tâche générée avec décomposition.

Décompositions en tests - Exemple

On veut tester une méthode qui compte le nombre de mots dans un document.

Compter le nombre de mots dans un document

Décompositions en tests - Exemple

On veut tester une méthode qui compte le nombre de mots dans un document.

~~Compter le nombre de mots dans un document~~
Effectuer une boucle sur chaque ligne
Compter le nombre des mots sur chaque ligne
Faire le total du décompte pour chaque ligne

Décompositions en tests - Exemple

On veut tester une méthode qui compte le nombre de mots dans un document.

~~Compter le nombre de mots dans un document~~
Effectuer une boucle sur chaque ligne
~~Compter le nombre des mots sur chaque ligne~~
Faire le total du décompte pour chaque ligne
Compter comme un seul mot la chaîne aaaa
Compter comme deux mots la chaîne aa aa
Compter comme trois mots la chaîne aa aa aa

TDD en pratique – Conventions de nommage

Il existe de nombreuses conventions de nommage :

1. MethodName_StateUnderTest_ExpectedBehavior
2. MethodName_ExpectedBehavior_StateUnderTest
3. test[Feature being tested]
4. Feature to be tested
5. Should_ExpectedBehavior_When_StateUnderTest
6. When_StateUnderTest_Expect_ExpectedBehavior

Ce qui donne :

```
isAdult_AgeLessThan18_False() \\ 1.
isAdult_False_AgeLessThan18() \\ 2.
testIsNotAnAdultIfAgeLessThan18() \\ 3.
IsNotAnAdultIfAgeLessThan18() \\ 4.
Should_ThrowException_When_AgeLessThan18() \\ 5.
When_AgeLessThan18_Expect_isAdultAsFalse \\ 6.
```

Source : DZone Agile Zone

TDD en pratique – Structure d'un test

Nous allons utiliser le framework `JUnit` qui fait partie de la famille des `XUnit`

- cppUnit (c++)
- CUnit (C)
- SUnit (SmallTalk)
- JUnit (Java)
- ...

TDD en pratique – Structure d'un test

- **Les 3 'A'**
  - Acteur : L'entité qui participe au test
  - Action : L'action effectuée sur l'acteur (i.e. méthode appelée)
  - Assertion : La fonction ou macro qui vérifie l'état du système testé et lance une exception en cas de résultat incorrect.
- **Les annotations** : `JUnit` fonctionne avec des annotations au-dessus des méthodes pour strucuter les tests.

```
import org.junit.jupiter.api.*;

public class TestFibonacci{
  @Test
  public void testFibonacci0(){
    Fibonacci fib = new Fibonacci(); // Acteur
    int result = fib.compute(0);// Action
    assertEquals(0,result); // Assertion
  }
}
```

TDD en pratique – Exemple

Calculer la séquence de Fibonacci

TDD en pratique – Exemple

~~Calculer la séquence de Fibonacci~~

$\text{Fibonacci}_0 = 0$

TDD en pratique – Exemple

~~Calculer la séquence de Fibonacci~~

$\text{Fibonacci}_0 = 0$

$\text{Fibonacci}_1 = 1$

TDD en pratique – Exemple

~~Calculer la séquence de Fibonacci~~

$\text{Fibonacci}_0 = 0$

$\text{Fibonacci}_1 = 1$

$\text{Fibonacci}_n = \text{Fibonacci}_{n\text{-}1} + \text{Fibonacci}_{n\text{-}2}$

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~

$Fibonacci_0 = 0$

$Fibonacci_1 = 1$

$Fibonacci_n = Fibonacci_{n-1} + Fibonacci_{n-2}$

Test

```
@Test
public void testFibonacci0(){
 // Acteur;
 Fibonacci fib = new Fibonacci();
 // Action
 int res = fib.compute(0);
 // Assertion
 assertEquals(0, res);
}
```

$\Rightarrow$

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~

$Fibonacci_0 = 0$

$Fibonacci_1 = 1$

$Fibonacci_n = Fibonacci_{n-1} + Fibonacci_{n-2}$

Test

```
@Test
public void testFibonacci0(){
 // Acteur;
 Fibonacci fib = new Fibonacci();
 // Action
 int res = fib.compute(0);
 // Assertion
 assertEquals(0, res);
}
```

$\Rightarrow$

Code

```
public class Fibonacci {
  public int compute(int n){
    return 0;
  }
}
```

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~

✓ **$\text{Fibonacci}_0 = 0$**

$\text{Fibonacci}_1 = 1$

$\text{Fibonacci}_n = \text{Fibonacci}_{n-1} + \text{Fibonacci}_{n-2}$

Test

```
@Test
public void testFibonacci0(){
 // Acteur;
 Fibonacci fib = new Fibonacci();
 // Action
 int res = fib.compute(0);
 // Assertion
 assertEquals(0, res);
}
```

$\Rightarrow$

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~
✓ $\text{Fibonacci}_0 = 0$
$\text{Fibonacci}_1 = 1$
$\text{Fibonacci}_n = \text{Fibonacci}_{n-1} + \text{Fibonacci}_{n-2}$

Test

```
@Test
public void testFibonacci01(){
 Fibonacci fib = new Fibonacci();
 assertEquals(0,fib.compute(0));
 assertEquals(1,fib.compute(1));
}
```

$\Rightarrow$

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~
✓ $\text{Fibonacci}_0 = 0$
$\text{Fibonacci}_1 = 1$
$\text{Fibonacci}_n = \text{Fibonacci}_{n\text{-}1} + \text{Fibonacci}_{n\text{-}2}$

Test

```
@Test
public void testFibonacci01(){
 Fibonacci fib = new Fibonacci();
 assertEquals(0,fib.compute(0));
 assertEquals(1,fib.compute(1));
}
```

$\Rightarrow$

Code

```
public class Fibonacci{
  public int compute(int n){
    if(n == 0) return 0;
    return 1;
  }
}
```

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~
✓ $\text{Fibonacci}_0 = 0$
✓ $\mathbf{\text{Fibonacci}_1 = 1}$
$\text{Fibonacci}_n = \text{Fibonacci}_{n\text{-}1} + \text{Fibonacci}_{n\text{-}2}$

Test

```
@Test
public void testFibonacci01(){
 Fibonacci fib = new Fibonacci();
 assertEquals(0,fib.compute(0));
 assertEquals(1,fib.compute(1));
}
```

$\Rightarrow$

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~
✓ $\text{Fibonacci}_0 = 0$
✓ $\text{Fibonacci}_1 = 1$
$\text{Fibonacci}_n = \text{Fibonacci}_{n-1} + \text{Fibonacci}_{n-2}$

Test

```
@Test
public void testFibonacciN(){
 // Acteur;
 Fibonacci fib = new Fibonacci();
 // Action + Assertion
 assertEquals(1,fib.compute(2));
 assertEquals(2,fib.compute(3));
 assertEquals(3,fib.compute(4));
 assertEquals(5,fib.compute(5));
 assertEquals(8,fib.compute(6));
}
```

⇒

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~

✓ $Fibonacci_0 = 0$

✓ $Fibonacci_1 = 1$

$Fibonacci_n = Fibonacci_{n-1} + Fibonacci_{n-2}$

Test

```
@Test
public void testFibonacciN(){
 // Acteur;
 Fibonacci fib = new Fibonacci();
 // Action + Assertion
 assertEquals(1,fib.compute(2));
 assertEquals(2,fib.compute(3));
 assertEquals(3,fib.compute(4));
 assertEquals(5,fib.compute(5));
 assertEquals(8,fib.compute(6));
}
```

⇒

Code

```
public class Fibonacci{
 public int compute(int n){
  if(n == 0) return 0;
  if(n <= 2) return 1;
  return this.compute(n-1) +
         this.compute(n-2);
 }
}
```

TDD en pratique - Exemple

~~Calculer la séquence de Fibonacci~~

✓ $Fibonacci_0 = 0$

✓ $Fibonacci_1 = 1$

✓ $\mathbf{Fibonacci_n = Fibonacci_{n-1} + Fibonacci_{n-2}}$

Test

```
@Test
public void testFibonacciN(){
 // Acteur;
 Fibonacci fib = new Fibonacci();
 // Action + Assertion
 assertEquals(1,fib.compute(2));
 assertEquals(2,fib.compute(3));
 assertEquals(3,fib.compute(4));
 assertEquals(5,fib.compute(5));
 assertEquals(8,fib.compute(6));
}
```

$\Rightarrow$

JUnit – Plus d'annotations

JUnit propose de nombreuses annotations pour structurer les tests. Nous avons déjà utilisé `@Test` mais il en existe d'autres

- `@BeforeAll` et `@AfterAll` : annote une méthode statique à appeler avant/après l'exécution de tous les tests. Souvent pour effectuer un setup/cleanup global (connexion à une DB, . . .).
- `@BeforeEach` et `@AfterEach` : annote une méthode à appeler avant/après chaque test (ouverture/création/supression de fichier, . . .).

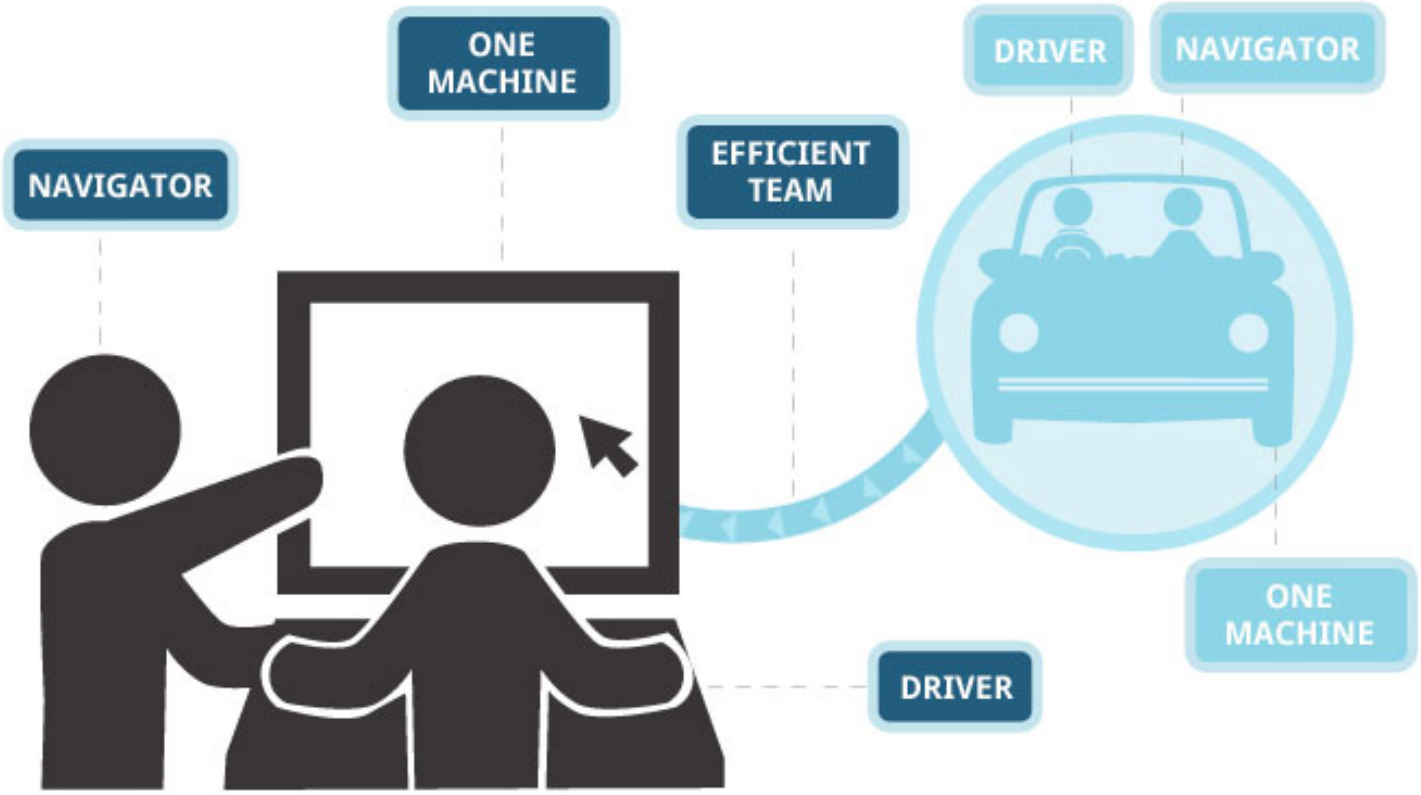
JUnit – Plus d'annotations

```java
import org.junit.jupiter.api.*;
public class JUnitTest {

  private Collection collection;

  @BeforeAll
  public static void oneTimeSetUp() {
    System.out.println("@BeforeAll - oneTimeSetUp");
  }
  @AfterAll
  public static void oneTimeTearDown() {
    System.out.println("@AfterAll - oneTimeTearDown");
  }
  @BeforeEach
  public void setUp() {
    collection = new ArrayList();
    System.out.println("@BeforeEach - setUp");
  }
  @AfterEach
  public void tearDown() {
    collection.clear();
    System.out.println("@AfterEach - tearDown");
  }
  @Test
  public void testEmptyCollection() {
    assertTrue(collection.isEmpty());
    System.out.println("@Test - testEmptyCollection");
  }
}
```

Pair programming



Source : Atalassian

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Pair programming

Rôles

- L'un écrit du code
- L'autre regarde et commente le code, pense à l'étape suivante, cherche de la documentation, pose des questions, ...

En pratique

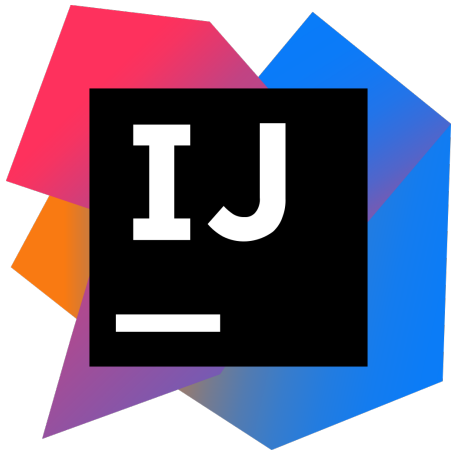
- En présentiel : à deux derrière un écran (ou deux pour effectuer des recherches en même temps)
- En distanciel : Teams & partage d'écran, ou des solutions plus avancées comme IntelliJ CodeWithMe et TMux/Git-based

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Intégration IDE

IDE - IntelliJ IDEA

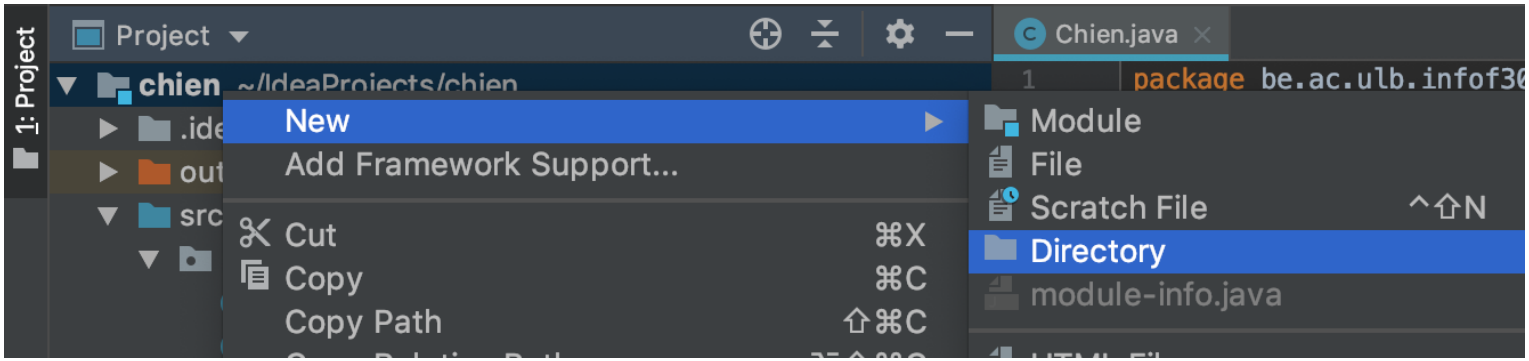


- Environnement de développement intégré : Éditeur de texte + Outils (Compilateur, l'éditeur de liens, débogueur) intégrés

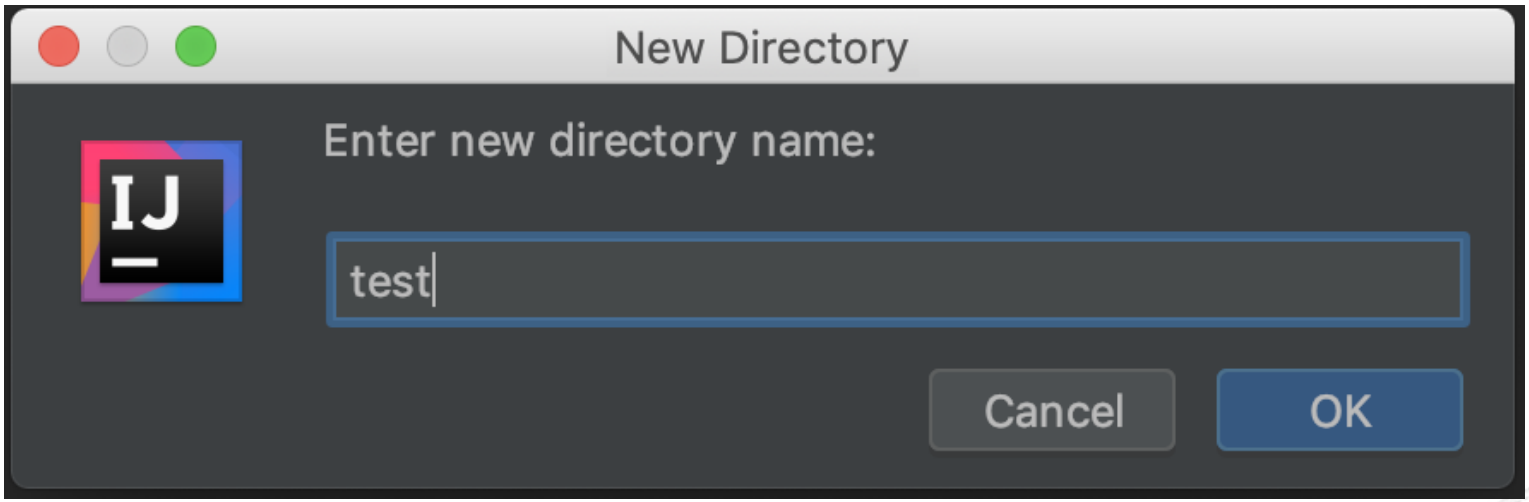
Nous utiliserons aussi :

- JUnit - Intégration du framework xUnit pour Java dans Intellij IDEA

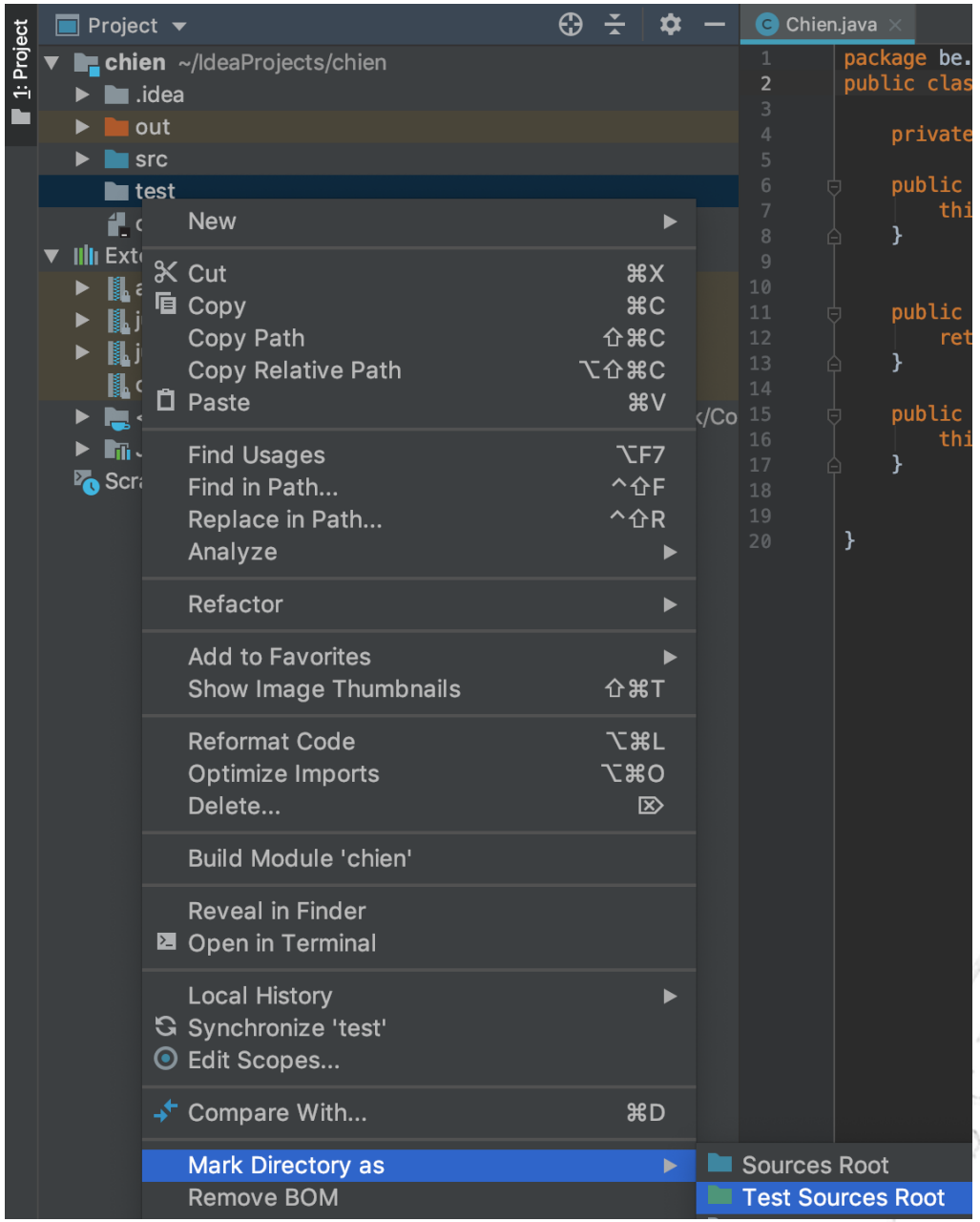
Création d'un nouveau test (JUnit) (1)



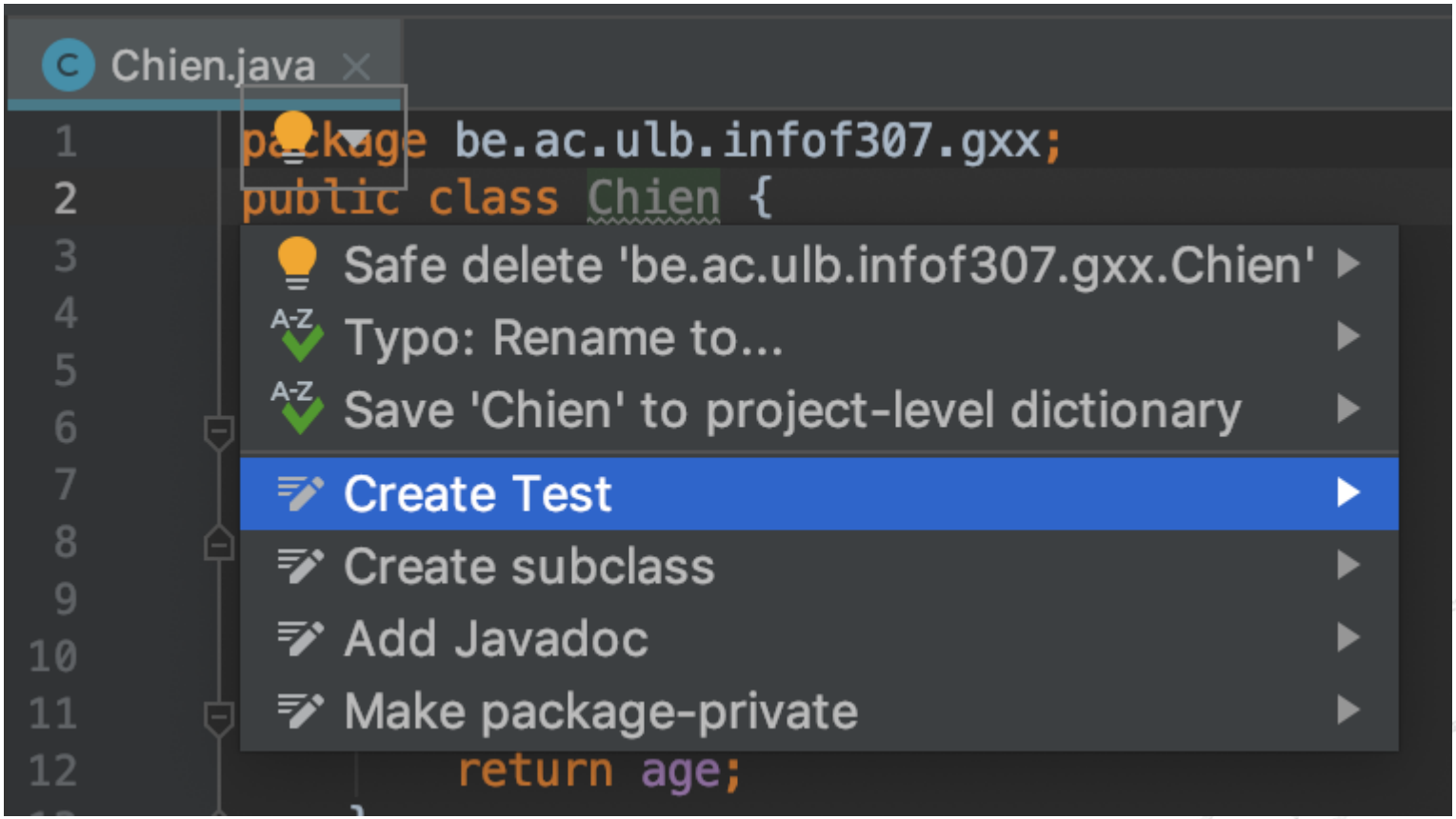
Création d'un nouveau test (JUnit) (2)



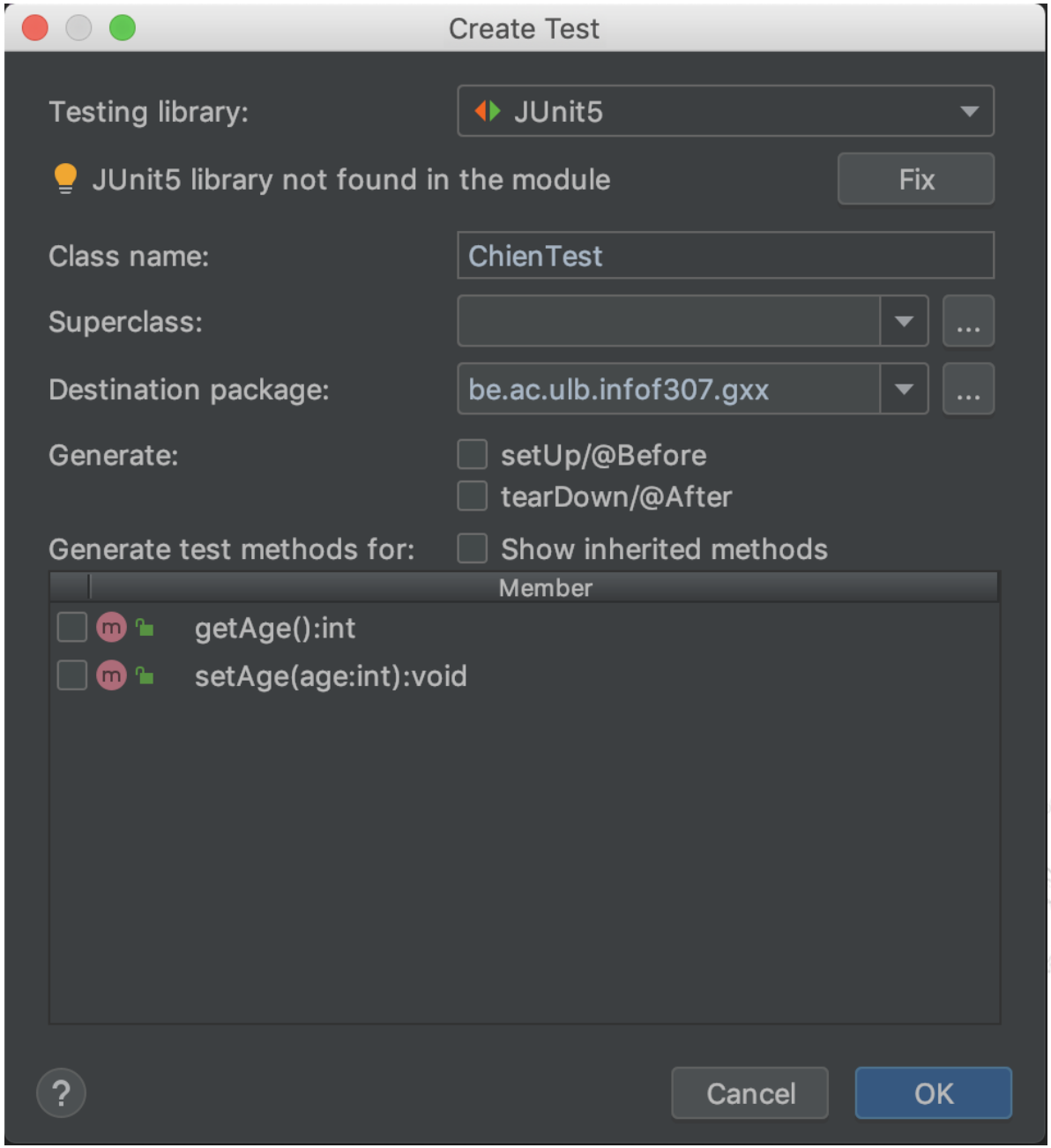
Création d'un nouveau test (JUnit) (3)



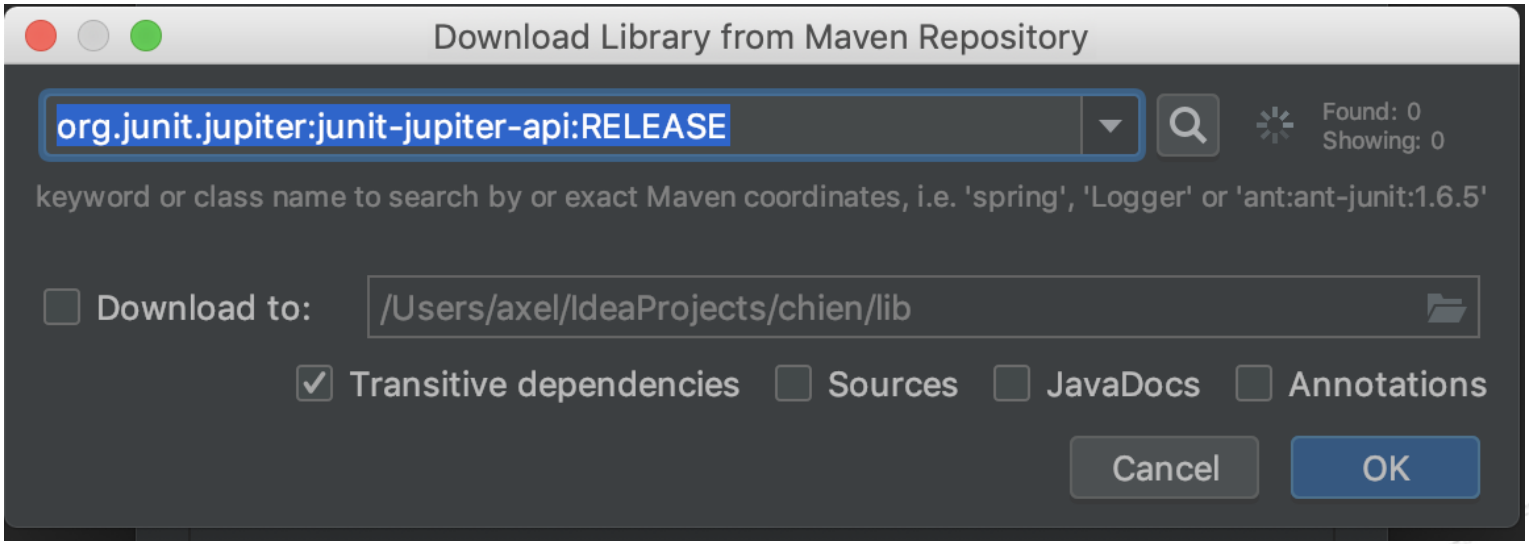
Création d'un nouveau test (JUnit) (4)



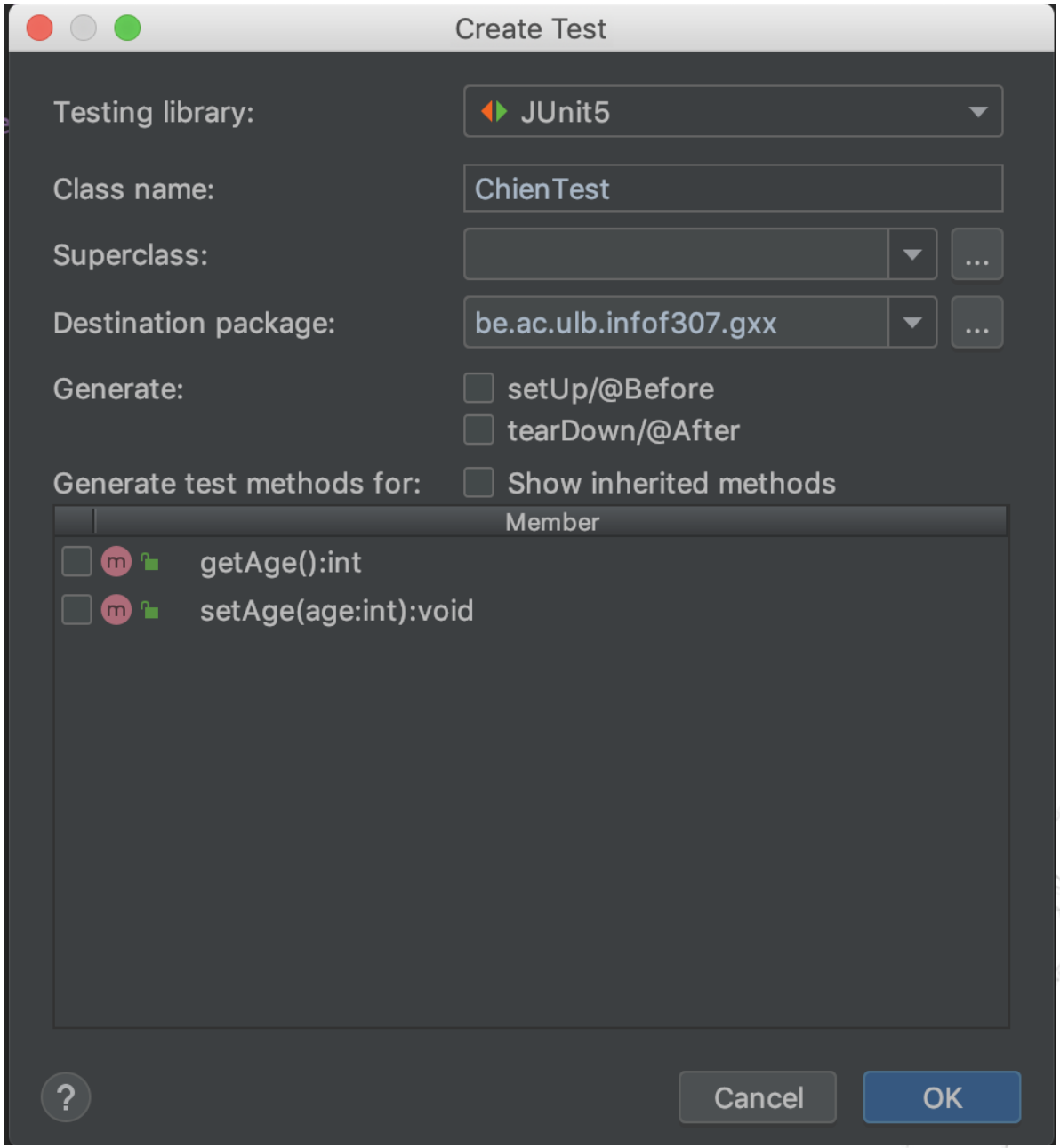
Création d'un nouveau test (JUnit) (5)



Création d'un nouveau test (JUnit) (6)



Création d'un nouveau test (JUnit) (7)



Ajout de la librairie JUnit

```
package be.ac.ulb.infof307.gxx;

import static org.junit.jupiter.api.Assertions.*;

class ChienTest {
```

Add library 'JUnit5.3' to classpath

Enable 'Settings | Editor | General | Auto Impo

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Exercices

Exercice 1

Implémenter, en utilisant les approches TDD et Pair Programming la méthode :

```
int divisible(int a, int b, int c)
```

qui retourne

- `b` si `a` est divisible par `b`
- `c` si `a` est divisible par `c`
- `b + c` si `a` est divisible par `b` et par `c`
- `test`, autrement.

L'exercice sera réalisé dans un dépôt Git, relié à un dépôt distant associé à votre compte GitHub.

ULB

UNIVERSITÉ LIBRE DE BRUXELLES

Exercice 2

Implémenter, en utilisant les approches TDD et Pair Programming, une classe Fraction qui permet de :

- Représenter une fraction
- Calculer la somme de deux fractions.
- Calculer la différence entre deux fractions.
- Calculer le produit de deux fractions.
- Calculer la division entre deux fractions.
- Calculer la réduction d'une fraction.

L'exercice sera réalisé dans un dépôt Git, relié à un dépôt distant associé à votre compte Gitlab.

Exercice 3

Implémenter, en utilisant les approches TDD et Pair Programming, une hiérarchie des classes qui permet de :

- Représenter un cercle, un triangle équilatéral et un carré.
- Calculer le périmètre des figures.
- Calculer l'aire des figures.

L'exercice sera réalisé dans un dépôt Git, relié à un dépôt distant associé à votre compte Gitlab.

Sources

- Exemple de la suite de Fibonacci : Beck, Kent. *Test-driven development : by example. Addison-Wesley Professional, 2003.*