

Test Driven Development (TDD)

Game jam!

Le TDD sert à réaliser des applications **complexes**, sinon, ça n'a pas de sense.

Alors, hop, 5 jours pour créer un jeu vidéo. Facile !

- Jour 1 = théorie, exemples et petits exercices
- Jour 2 et 3, on code en TDD
- Jour 4 et 5, on code en TDD + pair programming

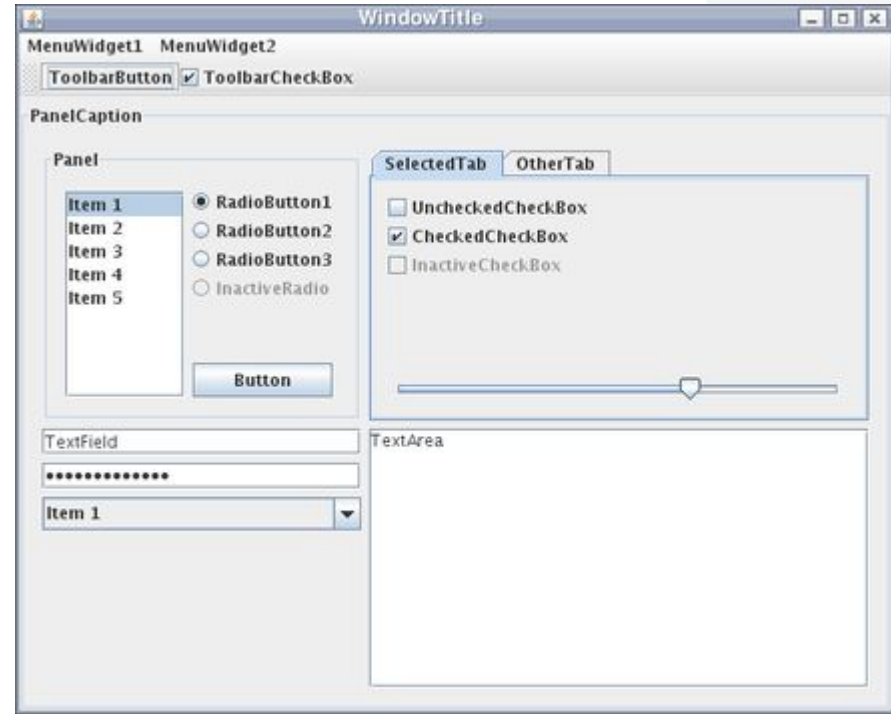
Un RPG au tour par tour

- => Le tour par tour est plus facile à coder sans moteur de jeu.
- => Le rendu peut être affiché via une console et du text (à l'ancienne !)
- => Le RPG est un style complet avec plusieurs concepts qui interagissent entre eux, c'est un très bon exercice

L'imagination est le meilleur moteur graphique

Mais les plus motivés pourront ajouter un rendu via Swing :

```
Shadow t'attaque!  
Tu perd 120 pv!  
  
SONIC: 500 pv max  
PV ACTUEL: 380  
  
SHADOW: 500 pv max  
PV ACTUEL: 400  
  
Choisissez  
  
1 - Attaquer  
2 - Defendre  
3 - Guerir  
4 - Spin Dash MP 25
```



Le jeu en quelques mots

Le joueur contrôle un personnage.

- Tant que le personnage est en vie :
 - Le personnage commence à affronter un monstre
 - Tant que le monstre est en vie et que le joueur est en vie
 - [Le joueur utilise des objets]+
 - Le joueur lance une capacité
 - [Le monstre utilise des objets]+
 - Le monstre lance une capacité
 - Si le personnage est mort, fin.
 - Le personnage reçoit une récompense
 - [Le personnage monte de niveau]

C'est une game jam !

Réalisez les objectifs initiaux, puis allez plus loin !

- Ajoutez une narration et décrivez votre univers
- Ajoutez des mécaniques
- Ajoutez un visuel sympa
- Ajoutez des événements autres que les combats

Vendredi après-midi, on pourra tester les jeux des uns et des autres (et pour rappel, après, y'a bière).

Environnement de travail

- Java SE Development Kit 8 (versions plus anciennes possibles)
- Eclipse (n'importe quelle version)

Créer le projet

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name:

☒ Use default location

Location: [Browse...](#)

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE (currently 'jdk1.8.0_221') [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files


☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

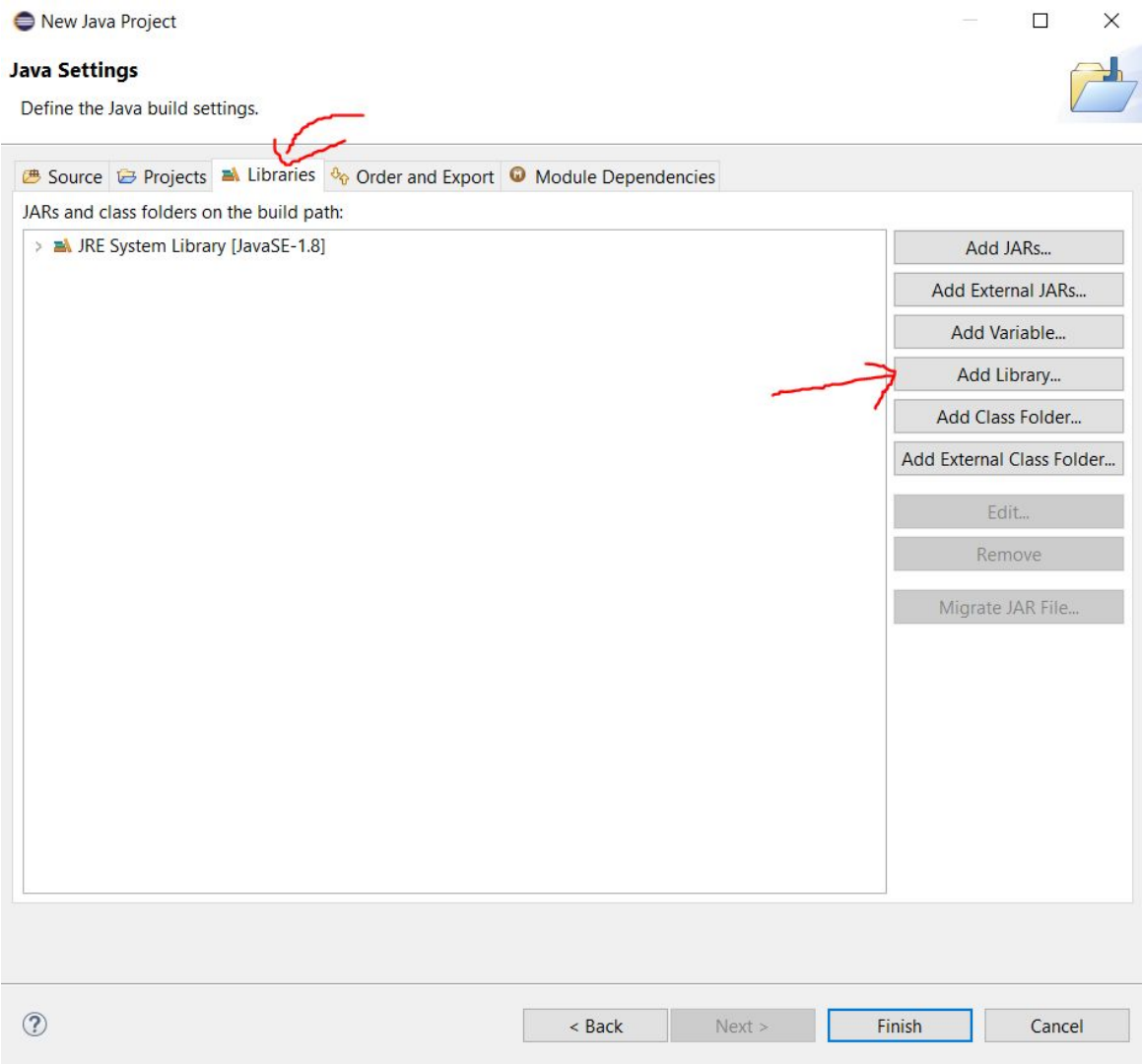
☐ Add project to working sets [New...](#)

Working sets: [Select...](#)

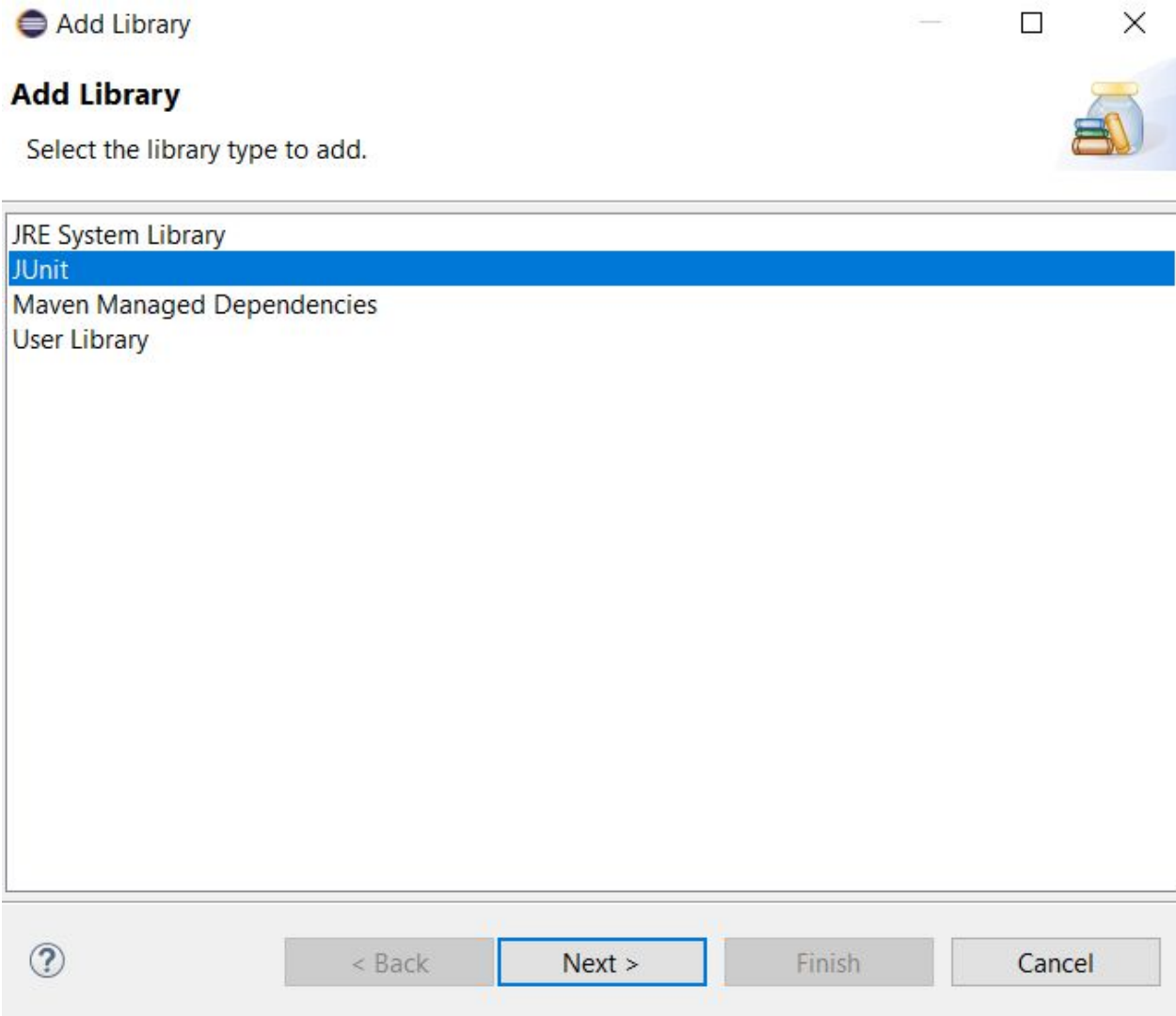
[?](#) [< Back](#) [Next >](#) [Finish](#) [Cancel](#)



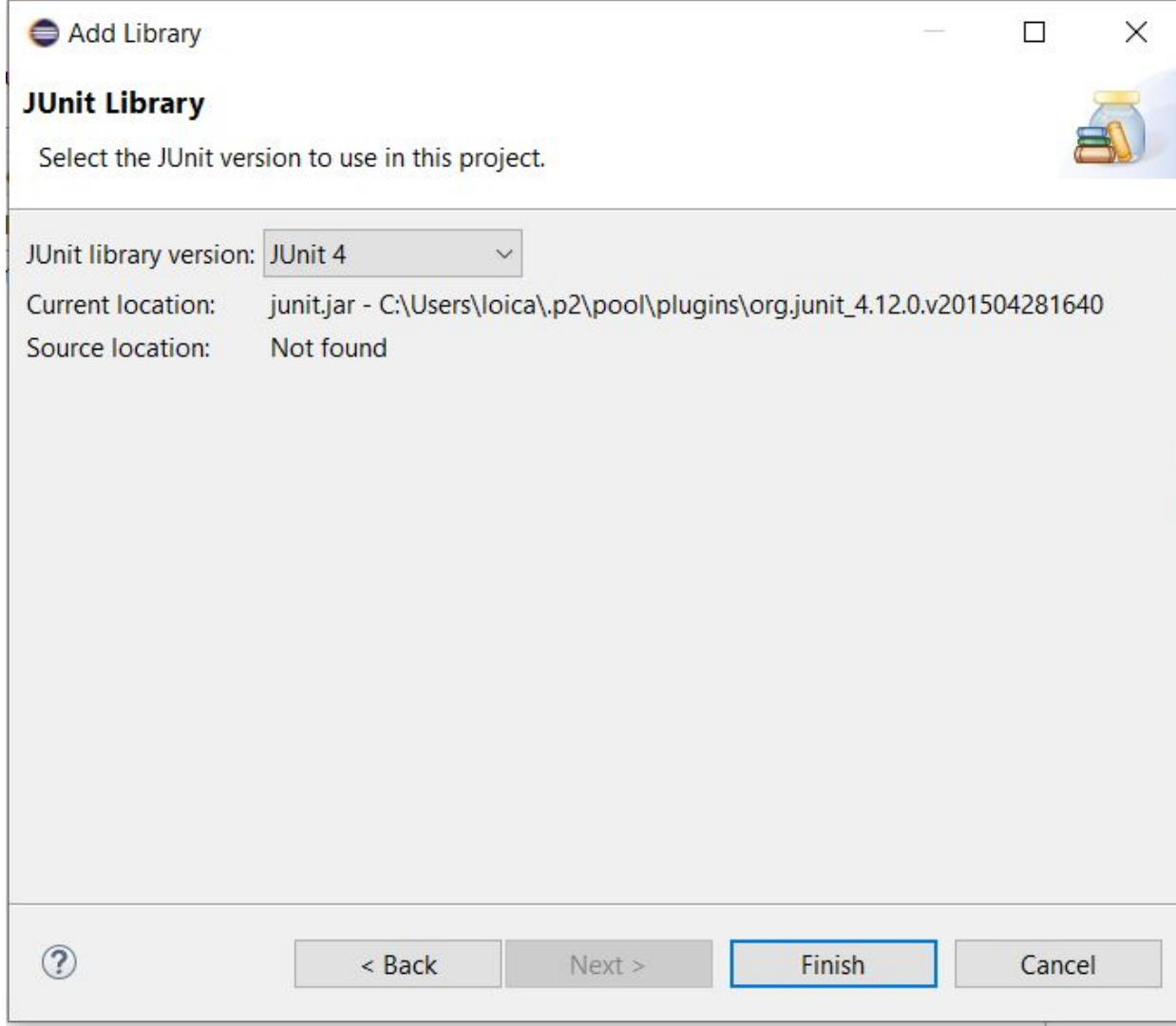
Créer le projet



Créer le projet



Créer le projet



(Bonus) Les dépendances dans la vraie vie

Nous avons importer une bibliothèque Java via Eclipse, “à la main”.

- En entreprise, vous utilisez sûrement Maven ;
- Maven est un outil de gestion de projets Java (théoriquement aussi valable pour d'autres langages).
- Toute la gestion du projet est écrite dans un fichier qui existe au sein du projet (le pom.xml).

Les dépendances dans la vraie vie

Nous avons importer une bibliothèque Java via Eclipse, “à la main”.

- N'importe qui peut checkout votre projet depuis le repo git et travailler directement sans rien avoir à configurer.
- Après tout, la configuration est déjà disponible au sein du pom.xml !

Maven

Create new POM

Maven POM

This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /My Project

Artifact

Group Id: MyProject

Artifact Id: MyProject

Version: 0.0.1-SNAPSHOT

Packaging: jar

Name: Projet TDD

Description: Projet réalisé en formation

?

Finish Cancel

```
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>MyProject</groupId>
5   <artifactId>MyProject</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>Projet TDD</name>
8   <description>Projet réalisé en formation</description>
9   <build>
10     <sourceDirectory>src</sourceDirectory>
11     <plugins>
12       <plugin>
13         <artifactId>maven-compiler-plugin</artifactId>
14         <version>3.8.0</version>
15         <configuration>
16           <source>1.8</source>
17           <target>1.8</target>
18         </configuration>
19       </plugin>
20     </plugins>
21   </build>
22 </project>
```

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
3   <modelVersion>4.0.0</modelVersion>
4   <groupId>MyProject</groupId>
5   <artifactId>MyProject</artifactId>
6   <version>0.0.1-SNAPSHOT</version>
7   <name>Projet TDD</name>
8   <description>Projet réalisé en formation</description>
9   <build>
10     <sourceDirectory>src</sourceDirectory>
11     <plugins>
12       <plugin>
13         <artifactId>maven-compiler-plugin</artifactId>
14         <version>3.8.0</version>
15         <configuration>
16           <source>1.8</source>
17           <target>1.8</target>
18         </configuration>
19       </plugin>
20     </plugins>
21   </build>
22 </project>
```


Dependencies

Dependencies



Add...

Remove

Properties...

Manage...

Dependencies

Select Dependency

Group Id: * org.mockito

Artifact Id: * mockito-all

Version: 1.9.5

Scope: test

Enter groupId, artifactId or sha1 prefix or pattern (*):

⚠ Index downloads are disabled, search results may be incomplete.

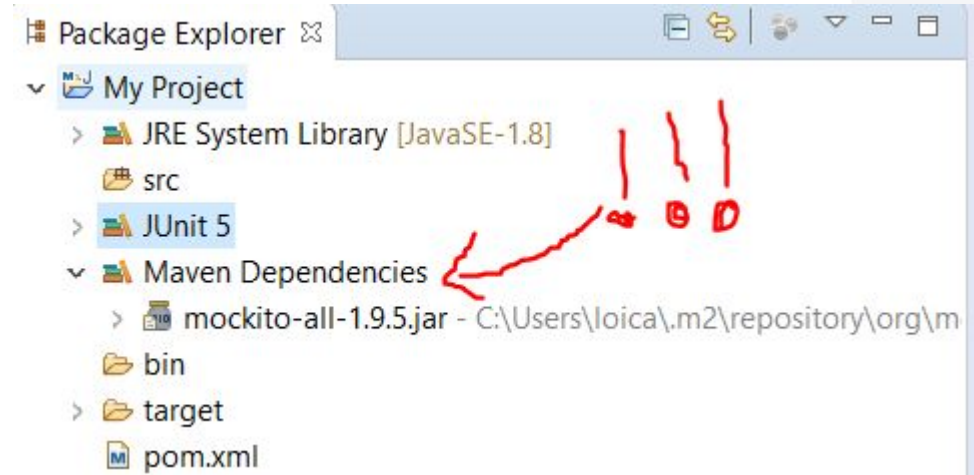
Search Results:



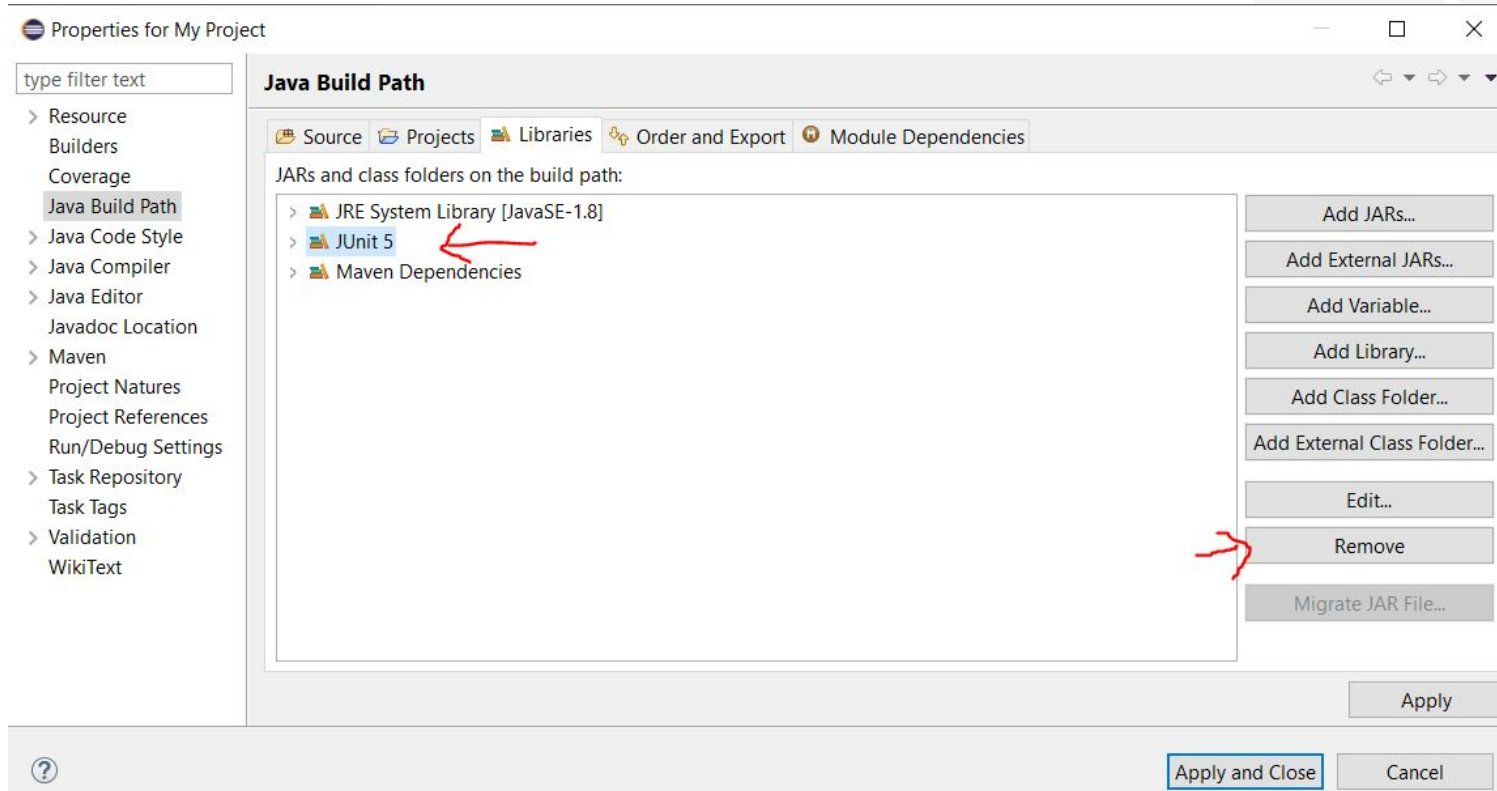
OK

Cancel

Maven, c'est le bien



Retirons la dépendance “à la main”



org.junit.jupiter:junit-jupiter:5.8.2



org.junit.jupiter:junit-jupiter: 5.8.2

[View on OSS Index](#)

[Browse](#)

[Downloads](#)

JUnit Jupiter (Aggregator)

Module "junit-jupiter" of JUnit 5.

Licenses [Eclipse Public License v2.0](#)

Home page <https://junit.org/junit5/>

Source code <https://github.com/junit-team/junit5>

Developers Stefan Bechtold <stefan.bechtold@me.com>
 Johannes Link <business@johanneslink.net>
 Marc Philipp <mail@marcphilipp.de>
 Matthias Merdes <matthias.merdes@heidelpay.com>
 Sam Brannen <sam@sambrannen.com>
 Christian Stein <sormuras@gmail.com>
 Juliette de Rancourt <derancourt.juliette@gmail.com>



org.junit.jupiter:junit-jupiter



Apache Maven

maven.apache.org



```
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter</artifactId>
  <version>5.8.2</version>
</dependency>
```



Gradle Groovy DSL

gradle.org



```
implementation 'org.junit.jupiter:junit-jupiter:5
<
>
```

Dependencies

Dependencies

mockito-all : 1.9.5 [test]

Add...

Remove

Properties...

Manage...

To manage your transitive dependency exclusions, please use the [Dependency Hierarchy](#) page

Overview Dependencies Dependency Hierarchy Effective POM pom.xml

Select Dependency

Group Id: * junit

Artifact Id: * junit

Version: 4.12

Scope: test

Enter groupId, artifactId or sha1 prefix or pattern (*):

⚠ Index downloads are disabled, search results may be incomplete.

Search Results:



OK

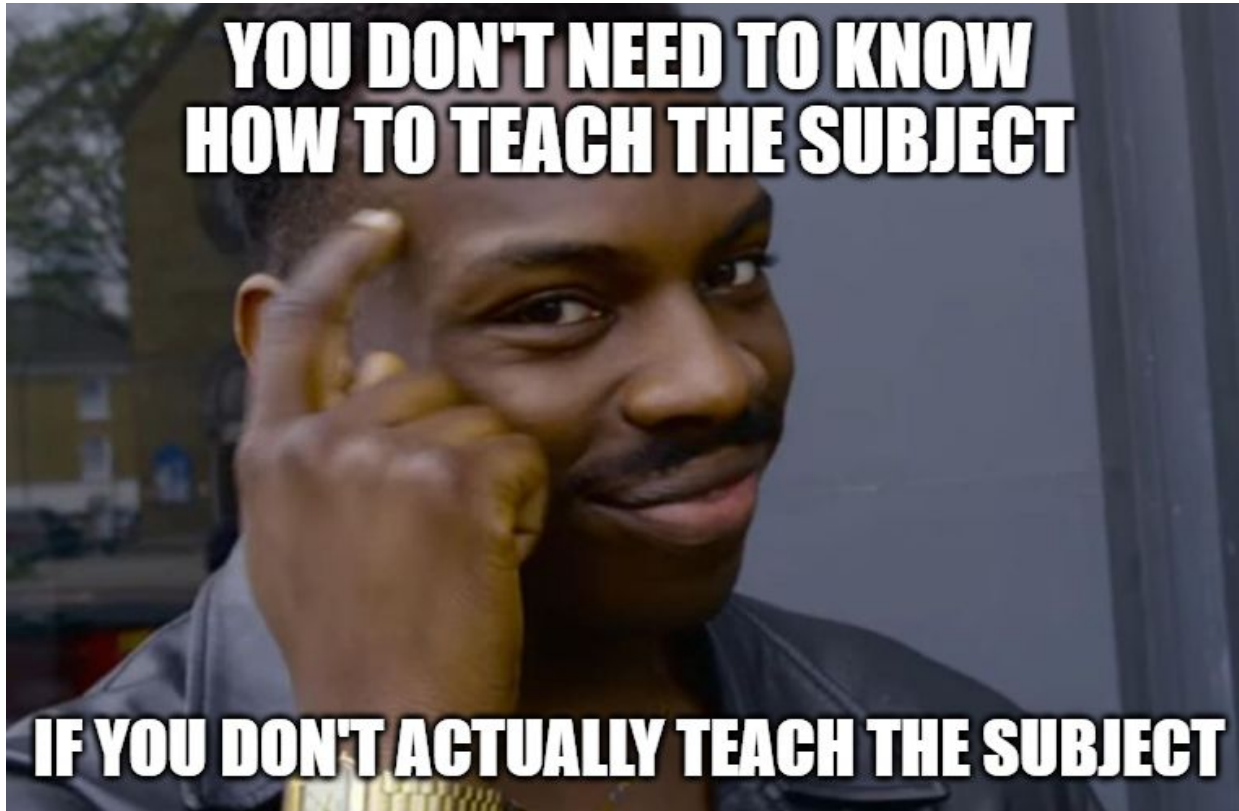
Cancel

Sous gradle

Ouvrir son fichier de configuration gradle.build :
Déclarer l'utilisation du repository de maven
Déclarer ses dépendances

```
repositories {  
    // Use Maven Central for resolving dependencies.  
    mavenCentral()  
}  
  
dependencies {  
    // Use JUnit Jupiter for testing.  
    testImplementation 'org.junit.jupiter:junit-jupiter:5.7.2'  
    testImplementation 'org.mockito:mockito-inline:4.6.1'  
    // This dependency is used by the application.  
    implementation 'com.google.guava:guava:30.1.1-jre'  
}
```

22 slides de hors-sujet plus tard



Un test “à la main”

On teste en ajoutant des “logs”.

En général, on le fait dans le logiciel lui-même.

C’est parfois compliqué de faire afficher le log.

Si on casse le code, on ne le verra pas tout de suite.

`public void nomDeMethode()`

=> Désolé, je suis trop habitué au C#

```
public class Character {  
  
    private int life;  
  
    public Character(int initialLife)  
    {  
        this.life = initialLife;  
    }  
  
    public boolean IsAlive()  
    {  
        System.out.println(life);  
        return life > 0;  
    }  
  
    //  
    public void Hit(int damage)  
    {  
        life -= damage;  
    }  
  
    public static void main(String[] args)  
    {  
        Character character = new Character(100);  
  
        character.Hit(10);  
        System.out.println(character.IsAlive());  
  
        character.Hit(90);  
        System.out.println(character.IsAlive());  
    }  
}
```


Dossier de test, dossier source

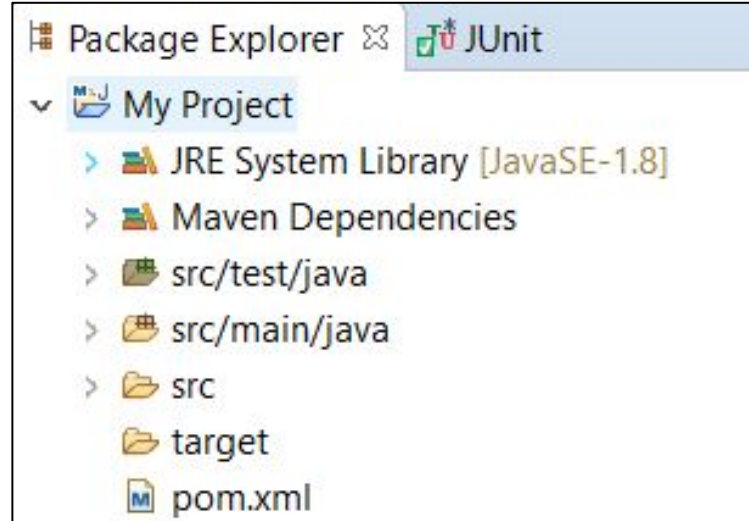
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:s
  <modelVersion>4.0.0</modelVersion>
  <groupId>MyProject</groupId>
  <artifactId>MyProject</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Projet TDD</name>
  <description>Projet réalisé en formation</description>

  <build>
    <sourceDirectory>src\main\java</sourceDirectory>
    <testSourceDirectory>src\test\java</testSourceDirectory>
```

Dossier de test, dossier source

- Mettre à jour le projet via Maven.
- Créer le dossier src/main/java
- Créer le dossier src/test/java

Should be good!



Sous gradle

La configuration du projet gradle est automatique.

On retrouvera donc nos dossiers test automatiquement.

Plus d'infos dans la mini-formation gradle ;)



Package Explorer JUnit

Finished after 0,022 seconds

Runs: 4/4 Errors: 0 Failures: 0

- game.CharacterTest [Runner: JUnit 4] (0,000 s)
 - TestHitWithDeath (0,000 s)
 - TestTwoHitsWithDeath (0,000 s)
 - TestHit (0,000 s)
 - TestTwoHitsWithoutDeath (0,000 s)

Failure Trace

My Project/pom.xml Character.java CharacterTest.java

```
1 package game;
2
3 import org.junit.Assert;
4 import org.junit.Test;
5
6 public class CharacterTest {
7
8     @Test
9     public void TestHit() {
10         Character character = new Character(10);
11
12         character.Hit(5);
13
14         Assert.assertTrue(character.IsAlive());
15     }
16
17     @Test
18     public void TestHitWithDeath() {
19         Character character = new Character(10);
20
21         character.Hit(10);
22
23         Assert.assertFalse(character.IsAlive());
24     }
25
26     @Test
27     public void TestTwoHitsWithDeath() {
28         Character character = new Character(10);
29
30         character.Hit(5);
31         character.Hit(5);
32         Assert.assertFalse(character.IsAlive());
33     }
34 }
```

Un test unitaire

Composition du test unitaire

- Classe
- Set-up / Tear-down
- 1 méthode = 1 test
 - Code du cas à jouer
 - Assert, instruction pour tester le résultat

Assert

Pour JUnit 3 & 4

org.junit
Class Assert

[java.lang.Object](#)
└ [org.junit.Assert](#)

public class **Assert**
extends [Object](#)

A set of assertion methods useful for writing tests. Only failed assertions are recorded. These methods can be used directly: `Assert.assertEquals(...)`, however, they read better if they are referenced through static import:

```
import static org.junit.Assert.*;  
...  
assertEquals(...);
```

See Also:
[AssertionError](#)

Constructor Summary

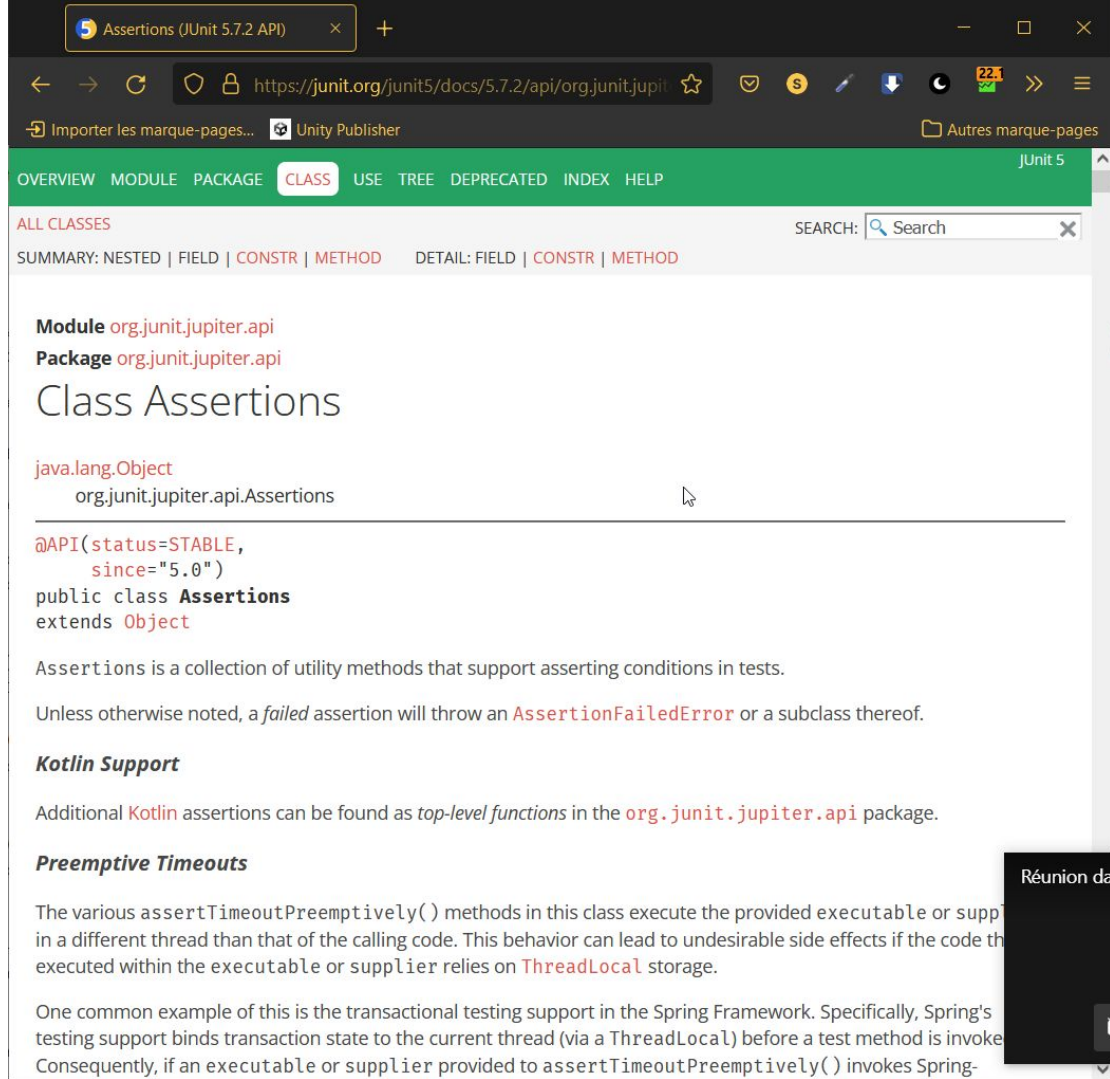
protected	Assert()
	Protect constructor since it is a static only class

Method Summary

static void	assertArrayEquals (byte[] expecteds, byte[] actuals) Asserts that two byte arrays are equal.
static void	assertArrayEquals (char[] expecteds, char[] actuals) Asserts that two char arrays are equal.
static void	assertArrayEquals (double[] expecteds, double[] actuals, double delta) Asserts that two double arrays are equal.
static void	assertArrayEquals (float[] expecteds, float[] actuals, float delta) Asserts that two float arrays are equal.
static void	assertArrayEquals (int[] expecteds, int[] actuals) Asserts that two int arrays are equal.
static void	assertArrayEquals (long[] expecteds, long[] actuals)

Assertion

Pour JUnit 5



The screenshot shows the JUnit 5 API documentation page for the `Assertions` class. The browser address bar shows the URL `https://junit.org/junit5/docs/5.7.2/api/org.junit.jupiter.api`. The page has a green header with navigation links: OVERVIEW, MODULE, PACKAGE, CLASS (selected), USE, TREE, DEPRECATED, INDEX, and HELP. Below the header, there's a search bar and a list of tabs: SUMMARY, NESTED, FIELD, CONSTR, METHOD, and DETAIL. The current view is the CLASS page. The class is `org.junit.jupiter.api.Assertions`, which extends `java.lang.Object`. The code snippet shows the class declaration: `@API(status=STABLE, since="5.0") public class Assertions extends Object`. The description states that `Assertions` is a collection of utility methods that support asserting conditions in tests. It also mentions that unless otherwise noted, a failed assertion will throw an `AssertionFailedError` or a subclass thereof. There are sections for **Kotlin Support** and **Preemptive Timeouts**. The **Preemptive Timeouts** section explains that `assertTimeoutPreemptively()` methods execute the provided executable or supplier in a different thread than that of the calling code, which can lead to undesirable side effects if the code relies on `ThreadLocal` storage. It gives an example of this in the Spring Framework's transactional testing support.

Module `org.junit.jupiter.api`
Package `org.junit.jupiter.api`

Class Assertions

`java.lang.Object`
`org.junit.jupiter.api.Assertions`

```
@API(status=STABLE,  
      since="5.0")  
public class Assertions  
    extends Object
```

`Assertions` is a collection of utility methods that support asserting conditions in tests.

Unless otherwise noted, a *failed* assertion will throw an `AssertionFailedError` or a subclass thereof.

Kotlin Support

Additional `Kotlin` assertions can be found as *top-level functions* in the `org.junit.jupiter.api` package.

Preemptive Timeouts

The various `assertTimeoutPreemptively()` methods in this class execute the provided executable or supplier in a different thread than that of the calling code. This behavior can lead to undesirable side effects if the code the executed within the executable or supplier relies on `ThreadLocal` storage.

One common example of this is the transactional testing support in the Spring Framework. Specifically, Spring's testing support binds transaction state to the current thread (via a `ThreadLocal`) before a test method is invoked. Consequently, if an executable or supplier provided to `assertTimeoutPreemptively()` invokes Spring-

Les états d'un test

The screenshot displays an IDE interface with two main panels. The left panel, titled 'JUnit', shows the results of a test run. It indicates that the tests finished after 0,023 seconds, with 6 runs, 1 error, and 1 failure. A list of tests is shown, including 'TestHitWithDeath', 'TestTwoHitsWithDeath', 'TestFailing', 'TestHit', 'TestThrowingUnexpectedException', and 'TestTwoHitsWithoutDeath'. The 'TestFailing' test is highlighted in red, indicating a failure. The right panel shows the source code of the 'CharacterTest.java' file. The code includes imports for 'Character' and 'Assert'. It defines a 'Character' class with a 'Hit' method and an 'IsAlive' method. The 'CharacterTest' class contains several test methods: 'TestHitWithDeath', 'TestTwoHitsWithDeath', 'TestFailing', and 'TestThrowingUnexpectedException'. The 'TestFailing' method is highlighted in blue, showing an assertion failure: `Assert.assertTrue(false);`. The 'TestThrowingUnexpectedException' method is also shown, throwing a new 'Exception'.

Package Explorer JUnit

Finished after 0,023 seconds

Runs: 6/6 Errors: 1 Failures: 1

game.CharacterTest [Runner: JUnit 4] (0,001 s)

- TestHitWithDeath (0,000 s)
- TestTwoHitsWithDeath (0,000 s)
- TestFailing (0,001 s)
- TestHit (0,000 s)
- TestThrowingUnexpectedException (0,000 s)
- TestTwoHitsWithoutDeath (0,000 s)

Failure Trace

My Project/pom.xml Character.java CharacterTest.java

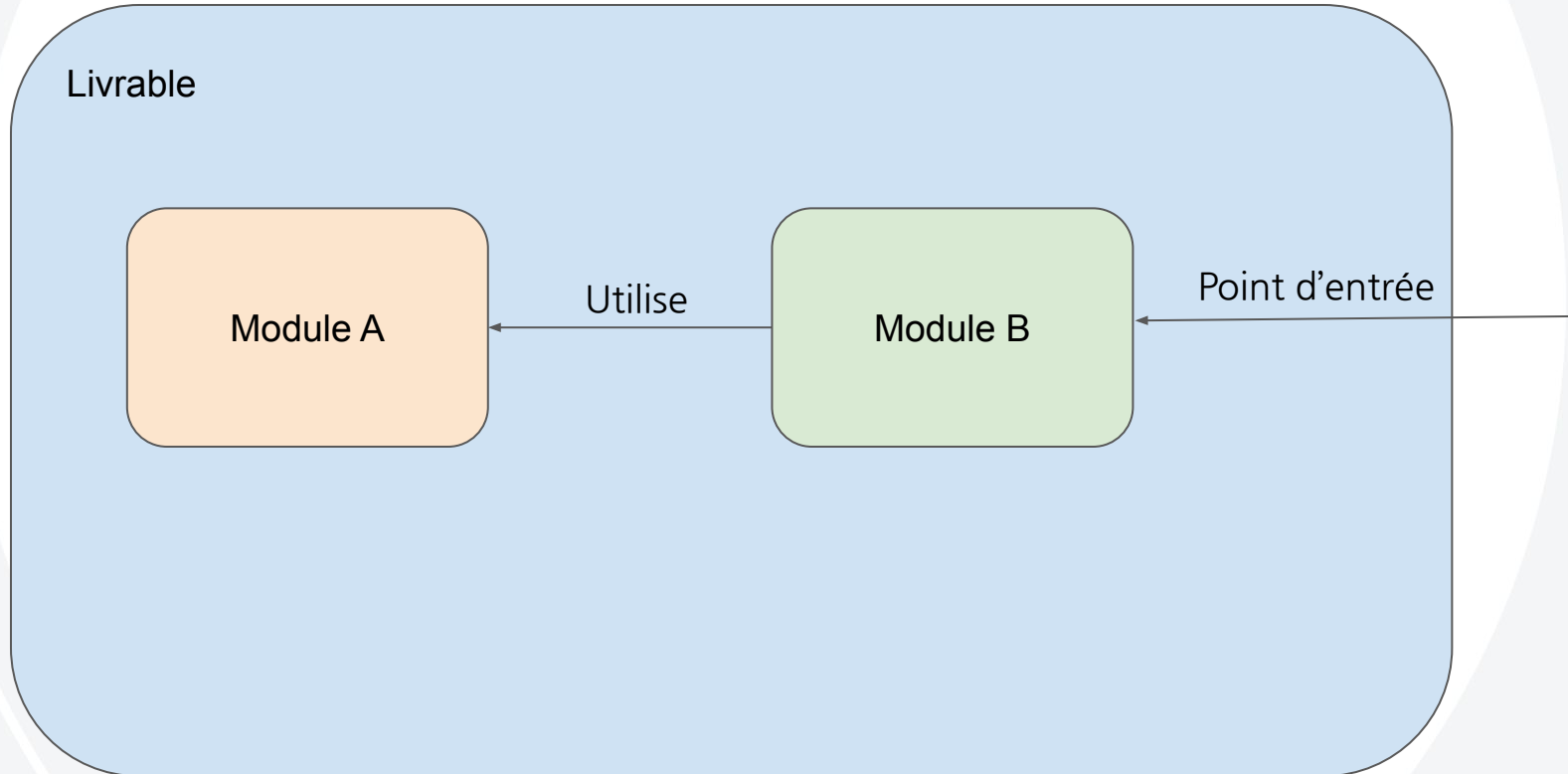
```
30 character.Hit(4);
31 character.Hit(6);
32
33 Assert.assertFalse(character.IsAlive());
34 }
35
36 @Test
37 public void TestTwoHitsWithoutDeath() {
38     Character character = new Character(10);
39
40     character.Hit(4);
41     character.Hit(5);
42
43     Assert.assertTrue(character.IsAlive());
44 }
45
46 @Test
47 public void TestFailing()
48 {
49     Assert.assertTrue(false);
50 }
51
52 @Test
53 public void TestThrowingUnexpectedException() throws Exception
54 {
55     throw new Exception();
56 }
57 }
58
```


Pourquoi des tests automatisés ?

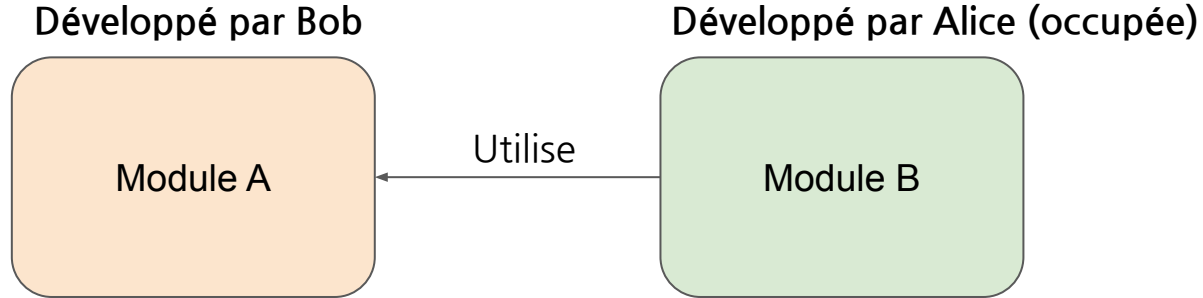
- Capacité de jouer tous les cas de figures rapidement
 - ... Sans en oublier
 - ... Sur l'ensemble du logiciel
-
- Test la validité du code vis-à-vis des **spécifications**
 - Diminue les risques de **régressions**

⇒ On a tous déjà réparé du code “qui marchait avant”.

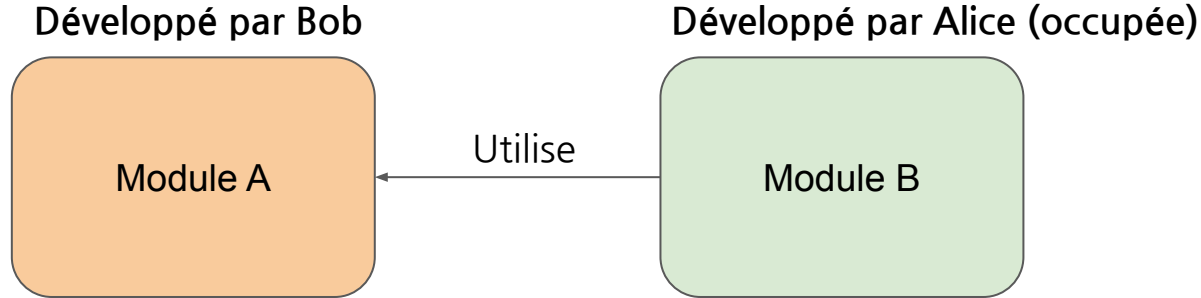
Pourquoi des tests automatisés ?



Pourquoi des tests automatisés ?

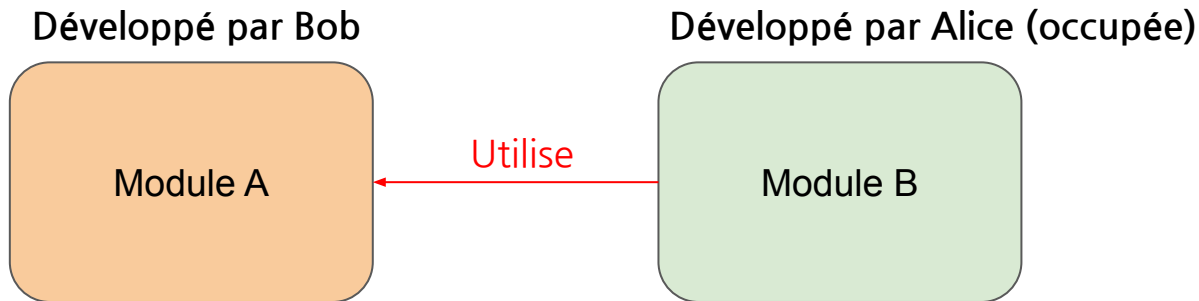


Pourquoi des tests automatisés ?



- Ajout de fonctionnalités
- Résolution des bugs

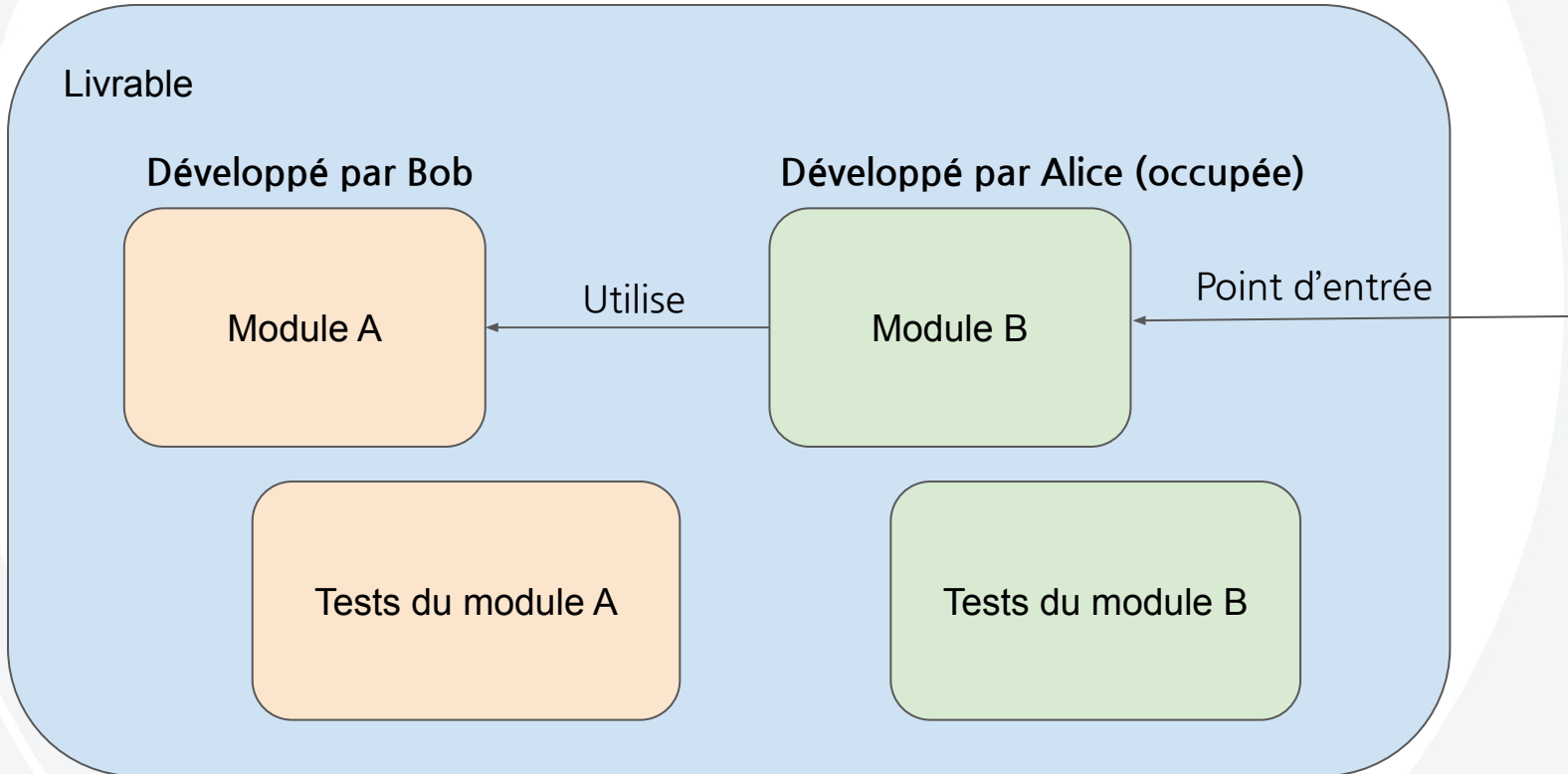
Pourquoi des tests automatisés ?



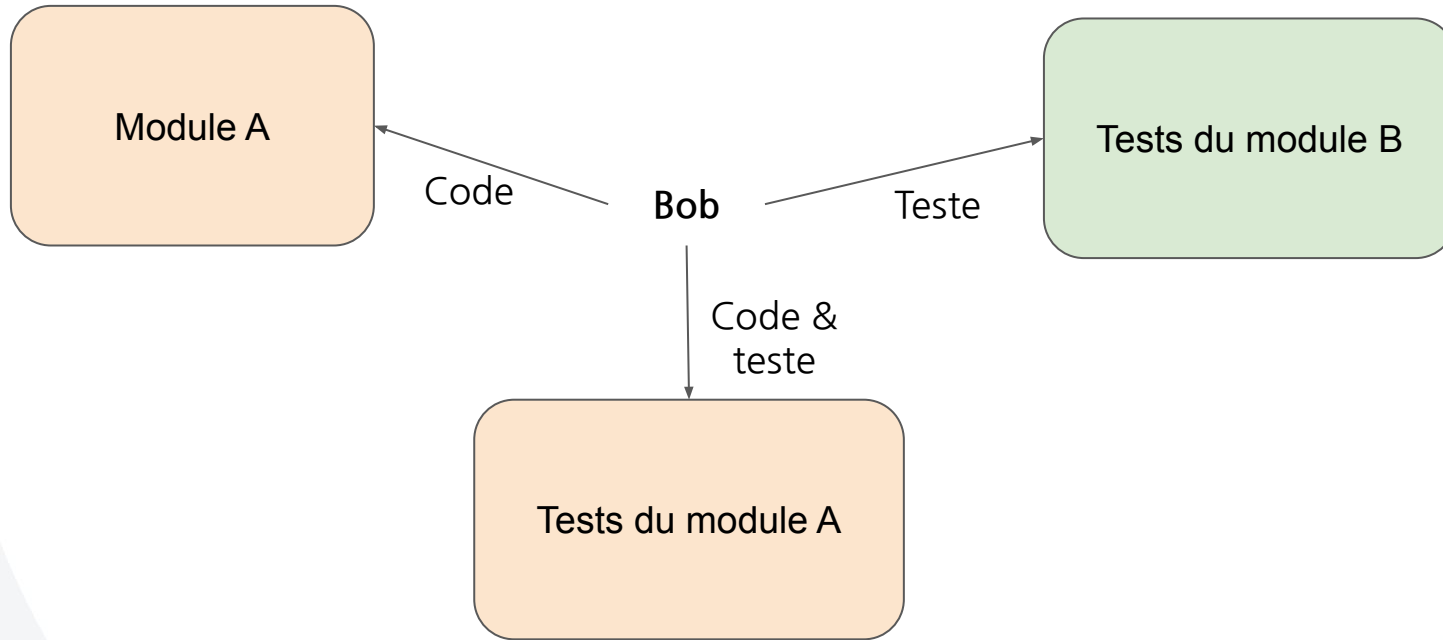
- Ajout de fonctionnalités
- Résolution des bugs

Comment Bob peut-il garantir de ne pas casser le code d'Alice ?

Pourquoi des tests automatisés ?

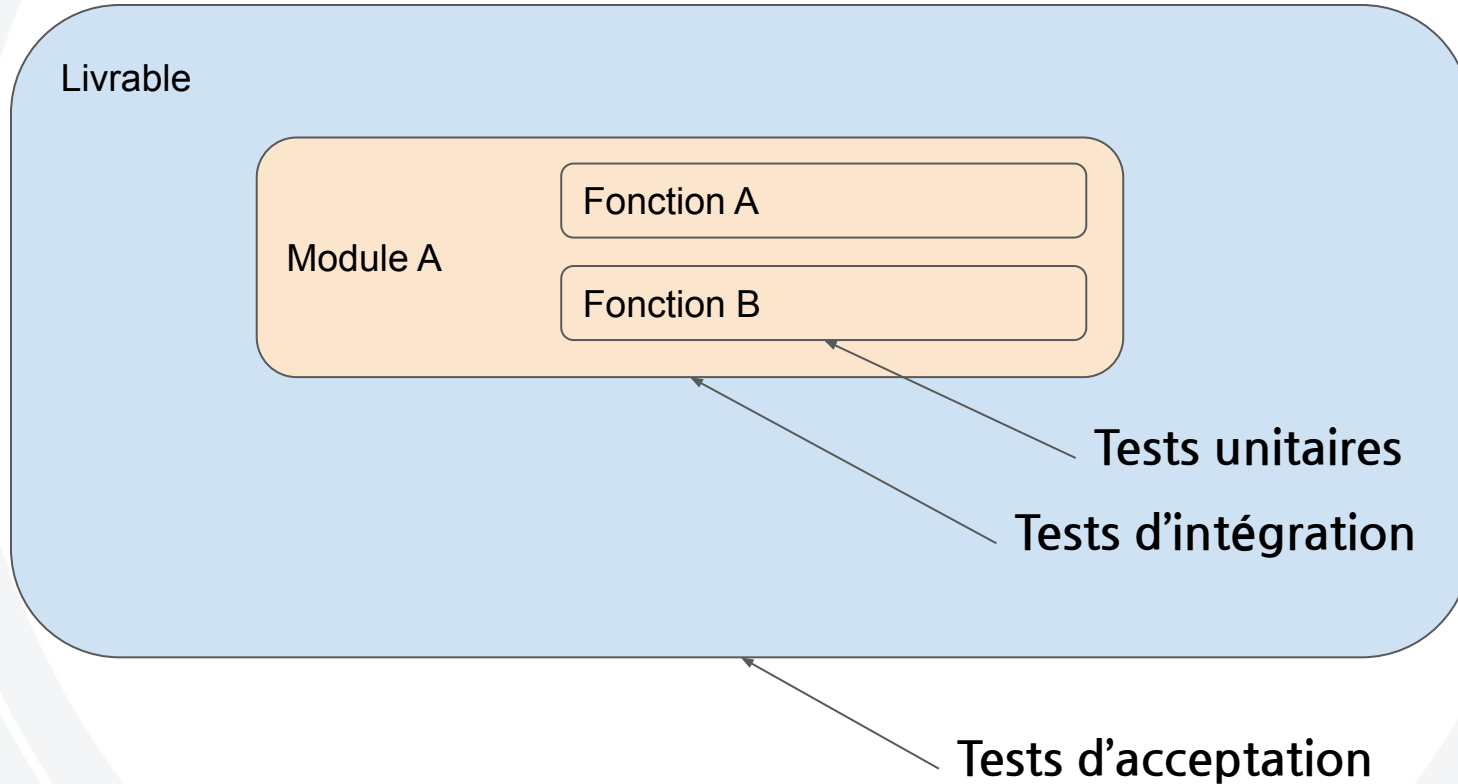


Pourquoi des tests automatisés ?

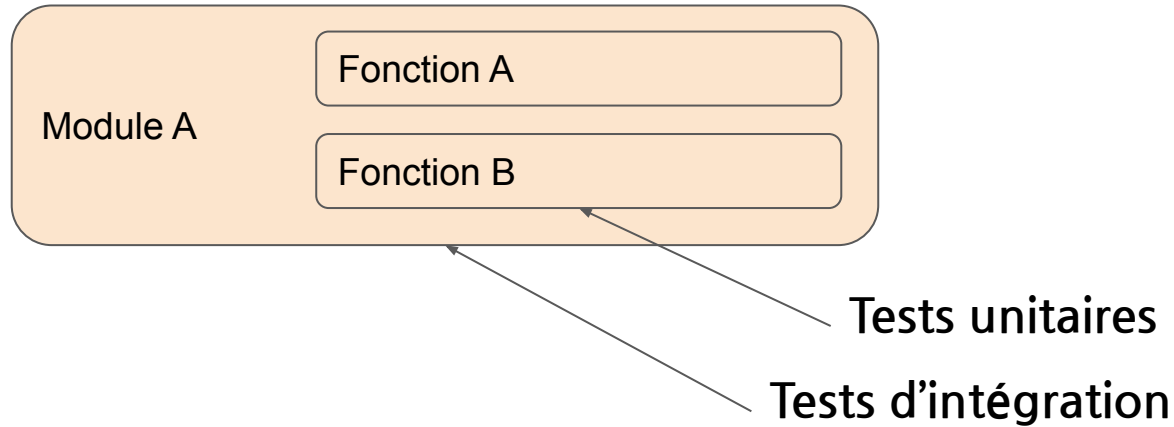


Alice peut garder l'esprit tranquille, Bob gère !

Les différents tests

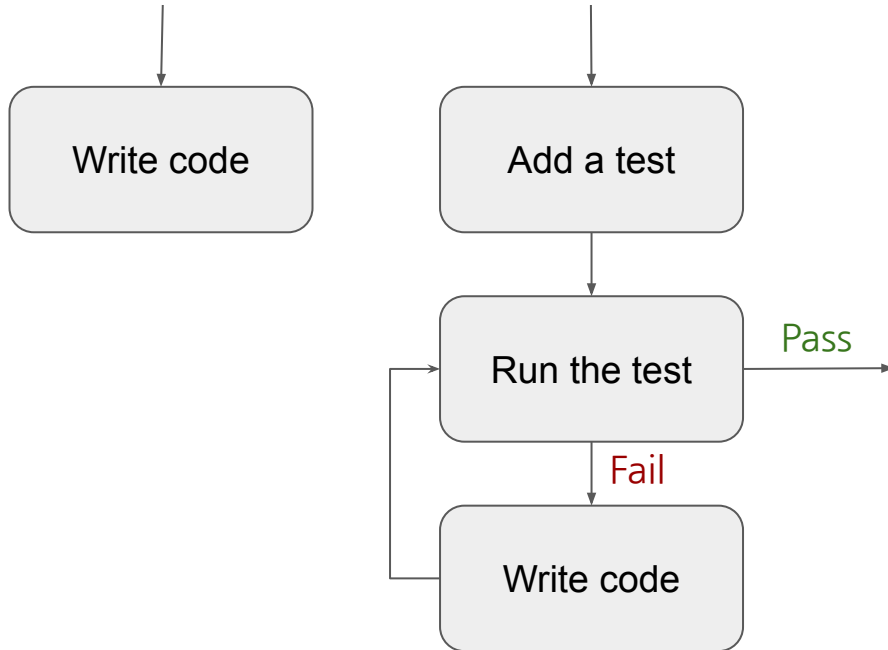


Les tests qui nous intéressent



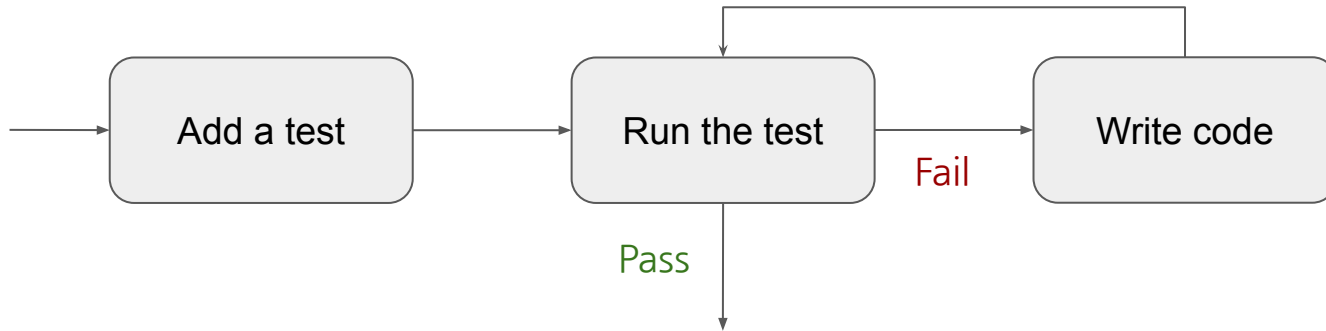
Méthode traditionnelle : on code d'abord

On code, puis plus tard, on écrit un test (peut-être).



Origines du TDD - “Test First Design”

Consiste à écrire un test avant de coder.



Test Driven Development

La version robuste du “Test First Design”.
C’est la même chose, mais avec :

- Une méthodologie
- Des bonnes pratiques
- Des design patterns

Première loi

“Vous devez écrire un test qui échoue avant d’écrire le code de production correspondant.”

L’objectif est de prouver qu’il existe du code à écrire.

=> Particulièrement utile en cas de bug, on prouve que le bug existe via un test.

=> Des fois, on a des surprises (le bug était une erreur de manip’ ou une incompréhension).

Première loi

“Vous devez écrire un test qui échoue avant d’écrire le code de production correspondant.”

- Vous devez établir votre API à l’avance (méthodes, arguments, retours)
- Vous pouvez écrire des méthodes vides pour avoir du code qui compile

Reprenons la classe Character

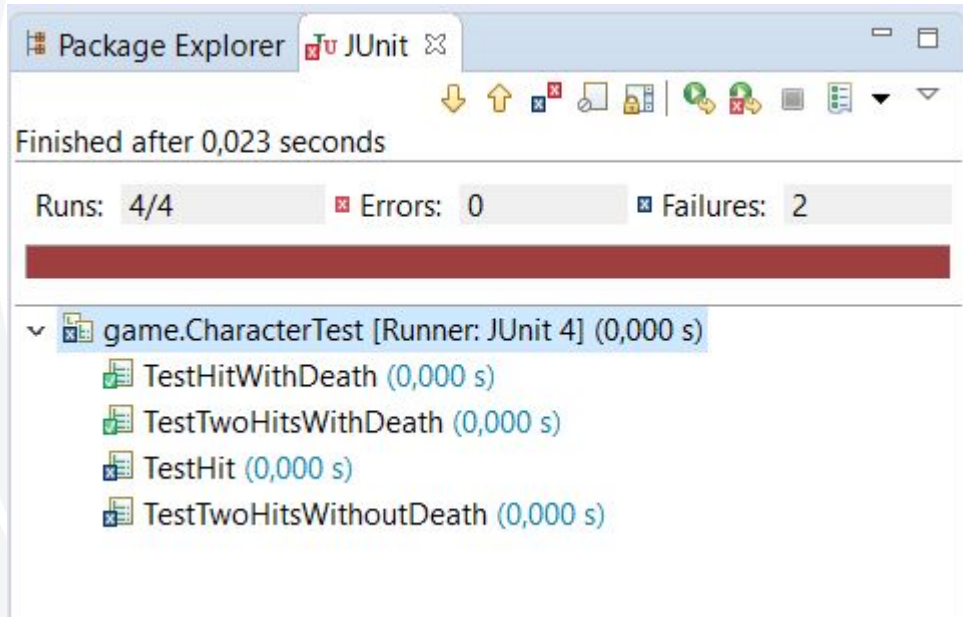
```
public class CharacterTest {  
    @Test  
    public void TestHit() {  
        Character character = new Character(10);  
  
        character.Hit(5);  
  
        Assert.assertTrue(character.IsAlive());  
    }  
}
```

Du code qui ne compile pas, ce n'est pas très intéressant.

Mais, nous avons pu écrire un code qui fait sens. C'est une façon d'écrire l'API !

Reprenons la classe Character

On écrit le code minimal pour que ça compile.



```
public class Character {  
  
    private int life;  
  
    public Character(int initialLife)  
    {  
        this.life = initialLife;  
    }  
  
    public boolean IsAlive()  
    {  
        return false;  
    }  
  
    public void Hit(int damage)  
    {  
  
    }  
}
```


Deuxième loi

“Vous devez écrire une seule assertion à la fois, qui fait échouer le test, ou qui échoue à la compilation.”

- On teste un aspect du code à la fois
- Au premier assert faux, le test s'arrête et les autres assertions du test ne sont pas évalués
- Dans ce cas, le rapport de test est donc incomplet

Deuxième loi

“Vous devez écrire une seule assertion à la fois, qui fait échouer le test, ou qui échoue à la compilation.”

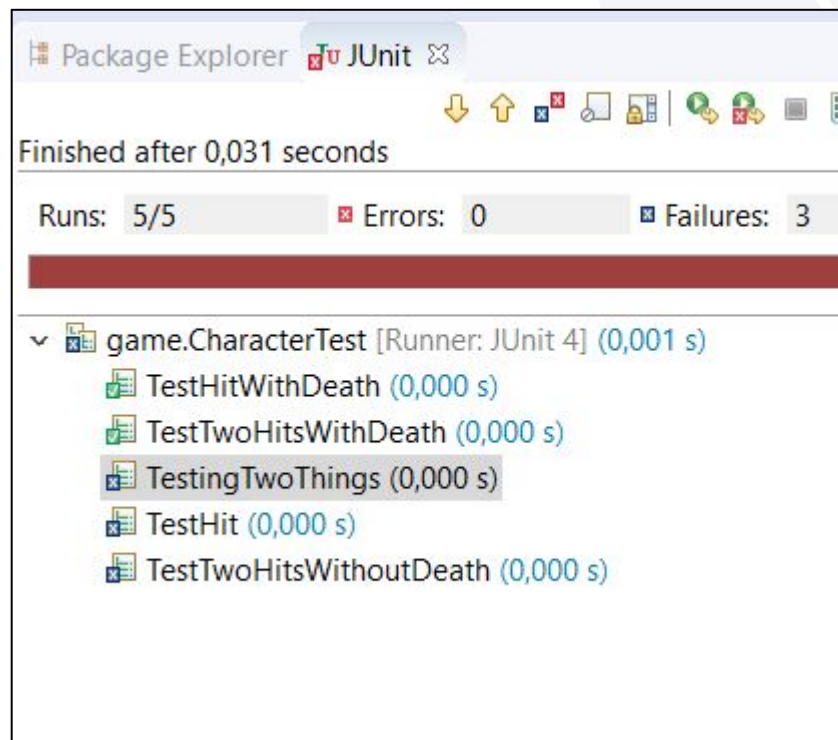
- ⇒ La bonne granularité des tests permet un meilleur **suivi de production**
- ⇒ Car coder des nouveautés, c’est aussi créer des régressions
- ⇒ Notamment en méthode Agile avec des cycles courts

Reprenons la classe character

```
@Test
public void TestingTwoThings()
{
    Character character = new Character(10);

    character.Hit(4);
    Assert.assertTrue(character.IsAlive());

    character.Hit(6);
    Assert.assertFalse(character.IsAlive());
}
```



The screenshot shows a JUnit test run results window. At the top, it says "Package Explorer" and "JUnit". Below that, it says "Finished after 0,031 seconds". The summary shows "Runs: 5/5", "Errors: 0", and "Failures: 3". The test results list includes:

- game.CharacterTest [Runner: JUnit 4] (0,001 s)
 - TestHitWithDeath (0,000 s)
 - TestTwoHitsWithDeath (0,000 s)
 - TestingTwoThings (0,000 s)
 - TestHit (0,000 s)
 - TestTwoHitsWithoutDeath (0,000 s)

Troisième loi

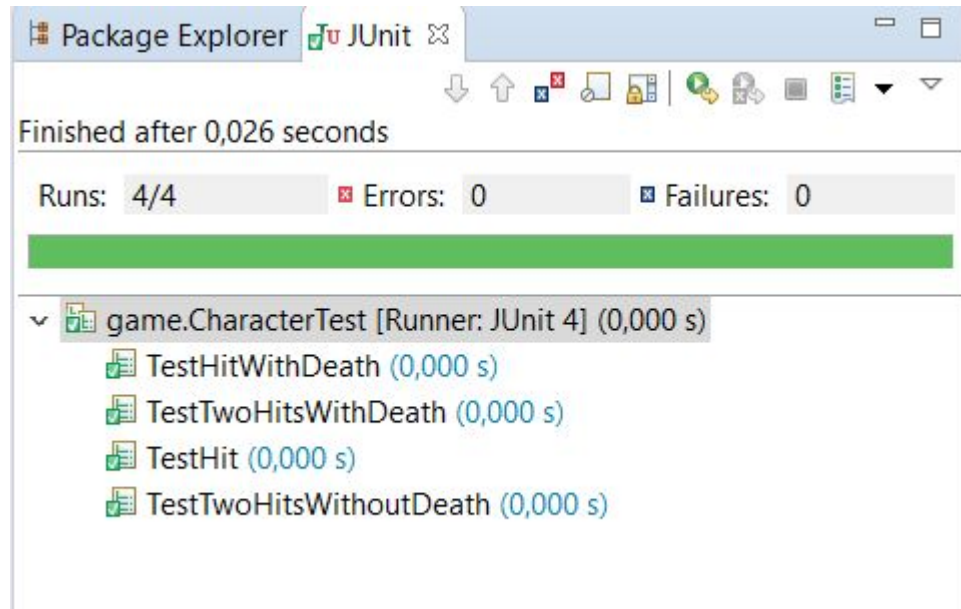
“Vous devez le minimum de code de production pour que l’assertion du test en échec soit satisfaite.”

=> On parle ici de périmètre fonctionnel

=> Si vous voulez ajouter une fonctionnalité, par exemple un argument supplémentaire, vous terminez d’abord votre cycle actuel

Reprenons la classe Character

Le but du jeu, écrire le minimum de code tel que :



TDD, la théorie

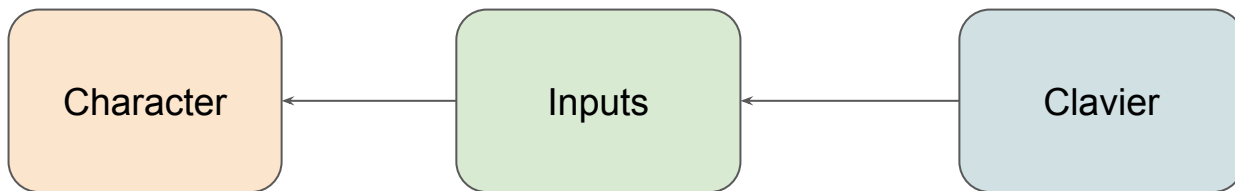
La méthodologie Test Driven Development à la lettre :

- Écrire un seul test, contenant une seule assertion
- Écrire le minimum de code pour faire passer le test
- Vérifier que tous les tests passent (le nouveau et les anciens)
- Remanier le code pour l'améliorer tout en gardant les tests au vert

En pratique, c'est impossible et **inefficace** de le faire “à la lettre” :)

Mocking

Comment tester du code avec des dépendances ?



Ici, nous souhaitons tester “Inputs”

Inputs

Si le joueur rentre la chaîne de caractère “1” au clavier, alors le personnage subit une attaque et reçoit 2 blessures.

Si le joueur rentre la chaîne de caractère “2” au clavier, alors on affiche si le personnage est en vie ou non.

Inputs, une classe compliquée à tester

```
public class InputsTest {  
    @Test  
    public void testAttack() {  
        Scanner scanner = new Scanner(System.in);  
        Character character = new Character(10);  
        Inputs inputs = new Inputs(scanner, character);  
  
        // How do we choose the input?  
        inputs.processNextInput();  
  
        // What should we test?  
    }  
}
```

Mockito

```
import org.mockito.Mockito;
import java.util.Scanner;

import org.junit.Test;

public class InputsTest {

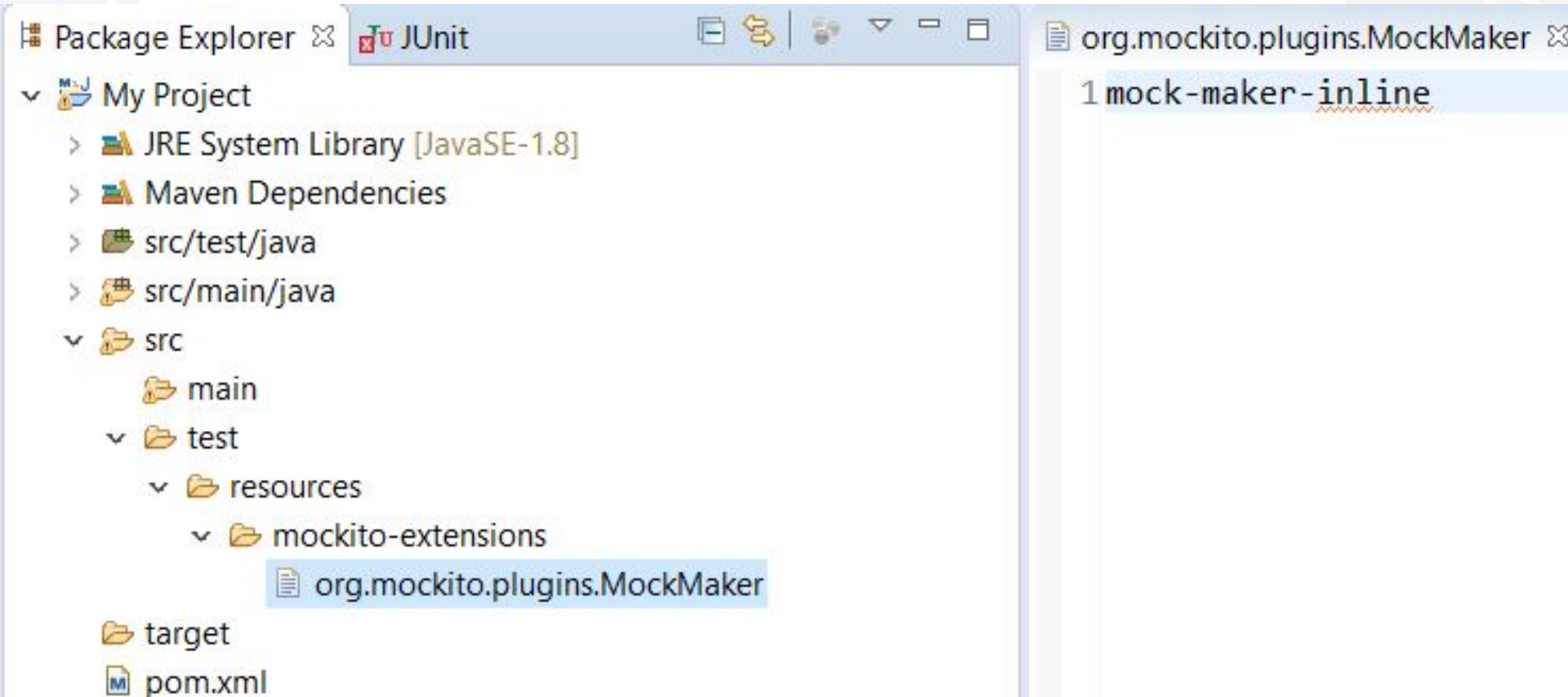
    @Test
    public void testAttack() {
        // Create a fake scanner
        Scanner scanner = Mockito.mock(Scanner.class);
        // That will return "1" when next() is called
        Mockito.when(scanner.next()).thenReturn("1");

        // Create a fake character
        Character character = Mockito.mock(Character.class);

        // The actual test code
        Inputs inputs = new Inputs(scanner, character);
        inputs.processNextInput();

        // Check that the character took 2 damage.
        Mockito.verify(character, Mockito.times(1)).hit(2);
    }
}
```

Encore une manip' ! (Franchement, Java...)



Package Explorer JUnit

Finished after 1,058 seconds

Runs: 1/1 Errors: 0 Failures:

testAttack [Runner: JUnit 4] (0,988 s)

Failure Trace

Wanted but not invoked:
character.hit(2);

-> at game.Character.hit(Character.java:19)

Actually, there were zero interactions with this mock.

at game.Character.hit(Character.java:19)

at game.InputsTest.testAttack(InputsTest.java:25)

```
import org.mockito.Mockito;
import java.util.Scanner;

import org.junit.Test;

public class InputsTest {

    @Test
    public void testAttack() {
        // Create a fake scanner
        Scanner scanner = Mockito.mock(Scanner.class);
        // That will return "1" when next() is called
        Mockito.when(scanner.next()).thenReturn("1");

        // Create a fake character
        Character character = Mockito.mock(Character.class);

        // The actual test code
        Inputs inputs = new Inputs(scanner, character);
        inputs.processNextInput();

        // Check that the character took 2 damage.
        Mockito.verify(character, Mockito.times(1)).hit(2);
    }
}
```

Limitations des mocks

Le test sera vrai car je l'ai mocké pour.

Mais fonctionne-t-il vraiment?

```
public void processNextInput()
{
    if (scanner.next() == "1")
        character.hit(10);
}
```

Spoiler alert: Non.



```
16 public void processNextInput()
17 {
18     String str = scanner.next();
19
20     if (str == "1")
21         character.hit(2);
22 }
23
24 public static void main(String[] args)
25 {
26     Scanner scanner = new Scanner(System.in);
27     Character character = new Character(2);
28
29     Inputs inputs = new Inputs(scanner, character);
30
31     inputs.processNextInput();
32     System.out.println("Character alive: " + character.isAlive());
33 }
34 }
```

Problems @ Javadoc Declaration Console

<terminated> Inputs [Java Application] C:\Program Files\Java\jdk1.8.0_221\bin\javaw.exe (15 sept. 2019)

```
1
Character alive: true
```


Egalité des références

```
public void processNextInput()
{
    if (scanner.next() == "1")
        character.hit(10);
}
```

Même référence

```
import org.mockito.Mockito;
import java.util.Scanner;

import org.junit.Test;

public class InputsTest {

    @Test
    public void testAttack() {
        // Create a fake scanner
        Scanner scanner = Mockito.mock(Scanner.class);
        // That will return "1" when next() is called
        Mockito.when(scanner.next()).thenReturn("1");

        // Create a fake character
        Character character = Mockito.mock(Character.class);

        // The actual test code
        Inputs inputs = new Inputs(scanner, character);
        inputs.processNextInput();

        // Check that the character took 2 damage.
        Mockito.verify(character, Mockito.times(1)).hit(2);
    }
}
```

```
16 public void processNextInput()
17 {
18     String str = scanner.next();
19
20     if (str.equals("1"))
21         character.hit(2);
22 }
23
24 public static void main(String[] args)
25 {
26     Scanner scanner = new Scanner(System.in);
27     Character character = new Character(2);
28
29     Inputs inputs = new Inputs(scanner, character);
30
31     inputs.processNextInput();
32     System.out.println("Character alive: " + character.isAlive());
33 }
34 }
```

Problems @ Javadoc Declaration Console

<terminated> Inputs [Java Application] C:\Program Files\Java\jdk1.8.0_221\bin\javaw.exe (15 sept. 2019

1

Character alive: false

Le mock est utile

Pouvoir forcer “Scanner” à renvoyer des inputs précises, c’est quand même pratique !

Simplement, restez vigilants !

En pratique, les tests d’intégrations et les tests de validations servent à détecter ce genre d’anomalie.

(Sinon, on en ferait pas)

Setup & Tear-down

Il est possible de “Set-Up” des opérations

Elles seront exécutées avant les tests

Soit avant TOUS les tests

Soit avant CHAQUE test

```
@BeforeAll
static void setup() {
    log.info("@BeforeAll - executes once before all test methods in this class");
}

@BeforeEach
void init() {
    log.info("@BeforeEach - executes before each test method in this class");
}
```

Setup & Tear-down

Il est possible de “Tear-Down” des opérations

Elles seront exécutées **après** les tests

Soit après **TOUS** les tests

Soit après **CHAQUE** test

```
@AfterEach
void tearDown() {
    log.info("@AfterEach - executed after each test method.");
}

@AfterAll
static void done() {
    log.info("@AfterAll - executed after all test methods.");
}
```

Setup & Tear-Down

Cela permet par exemple:

- D'initialiser un Scanner qui ne sera fermé qu'à la fin des tests pour pouvoir être utilisé tout le long.

- De mettre en place une connexion à la BDD pour qu'elle soit elle aussi utilisée tout le long des tests