FMS ACADEMY BY FMS

FACYLITIES MULTI SERVICES

Une entreprise ordinaire à l'agilité extraordinaire

FORMATION JAVA SPRING ANGULAR

JAVA UNIT TEST

mohamed.el-babili@fms-ea.com

33+ 628 111 476

Version : 2.0

DMAJ : 06/06/23

Module : DEV-JAVA-004
Maven & test unitaire

01 Connaître le processus de développement d'un projet avec Maven
02 Utiliser Maven pour injecter des dépendances
03 Utiliser Maven pour générer une application et autre fonctionnalités
04 Mise en œuvre de tests unitaires avec Junit & TDD

SOMMAIRE

Pourquoi tester une application ?

Pour répondre à des impératifs (d'agilité) :

- qualité des applications (bugs…)
- conformité aux attentes du client (spécification fonctionnelles)
- anticiper sur les cas d'utilisations et sur l'évolution d'une application
- contraintes techniques & délais de livraison (cout)
- confiance du client (éviter la régression)
- communication entre les développeurs

Comment tester une application ?

En réalisant nous même des **tests manuels** en passant par tous les chemins…

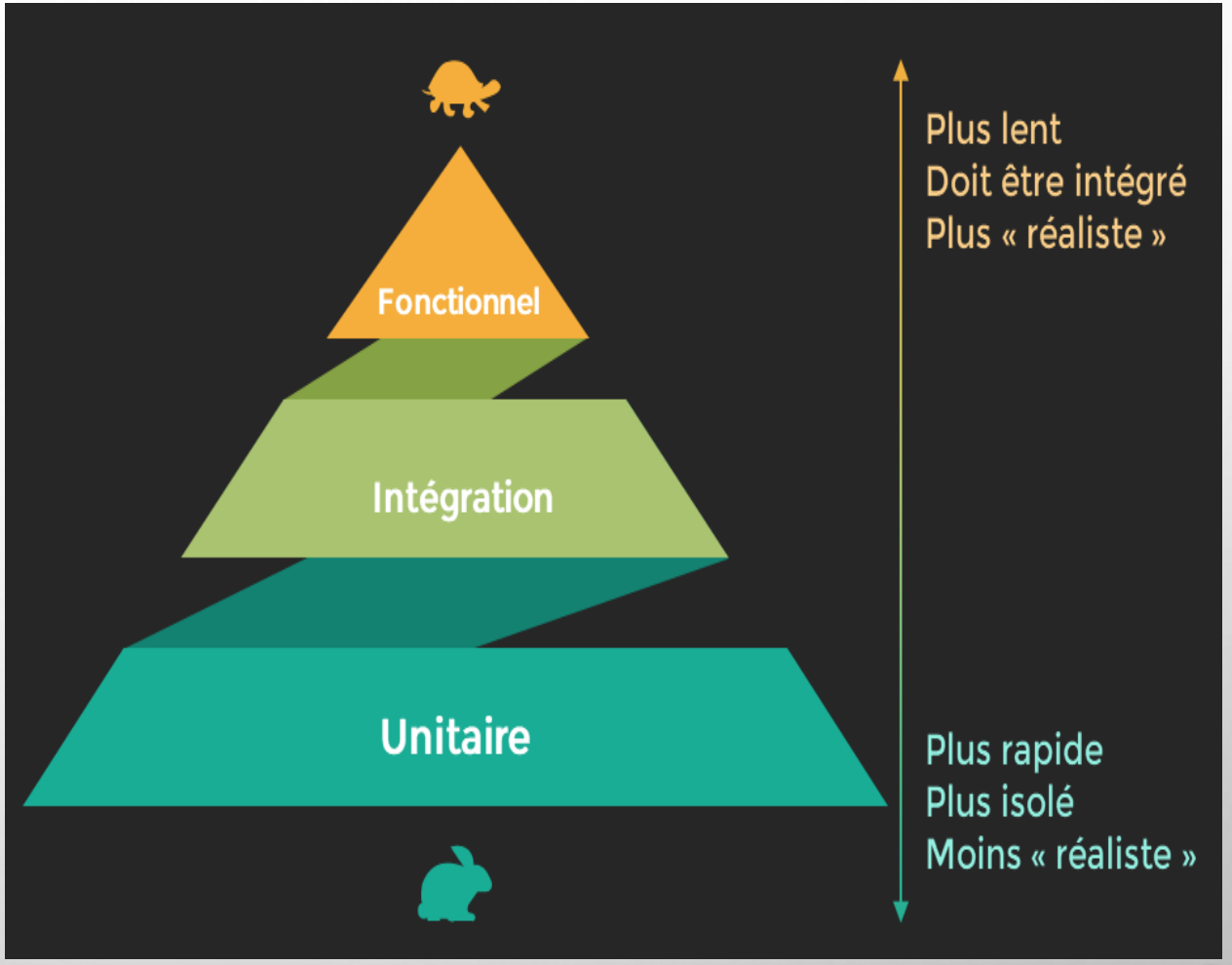
Le pb c'est qu'ils sont souvent incomplet voir insuffisant et ils ne tiennent pas toujours compte de l'évolution des applications.

En effet, sur des projets importants, nombreuses sont les interventions de devs, de versions en versions, est-ce qu'ils répètent tous les tests ?

En automatisant les tests, on garanti que tous les tests prévus sur chaque version d'une application seront effectués automatiquement

Différents types de tests

Mike Cohn, l'un des fondateurs du mouvement Agile, présente ces types de tests sous forme de **pyramide des tests.**



Les tests fonctionnelles (end to end) visent à simuler le comportement d'un User final.

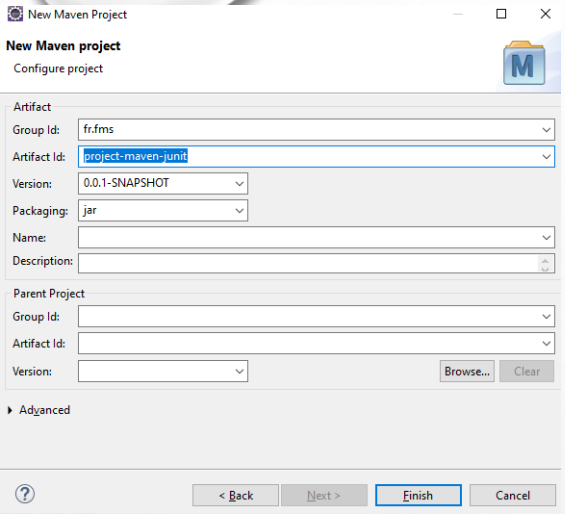
Les tests d'intégrations vérifient que les unités de code fonctionnent ensemble correctement comme prévu.
Par ex, tester l'inter action entre 2 couches

Les tests unitaires sont les plus nombreux et **les plus importants**. Ils garantissent que chaque unités de code ou fonctionnalités se comporte comme prévu.
Ils sont rapides, faciles à exécuter, stables quelle que soit les évolutions et rentables car une fois écrits, ils sont exécutés de nombreuses fois.

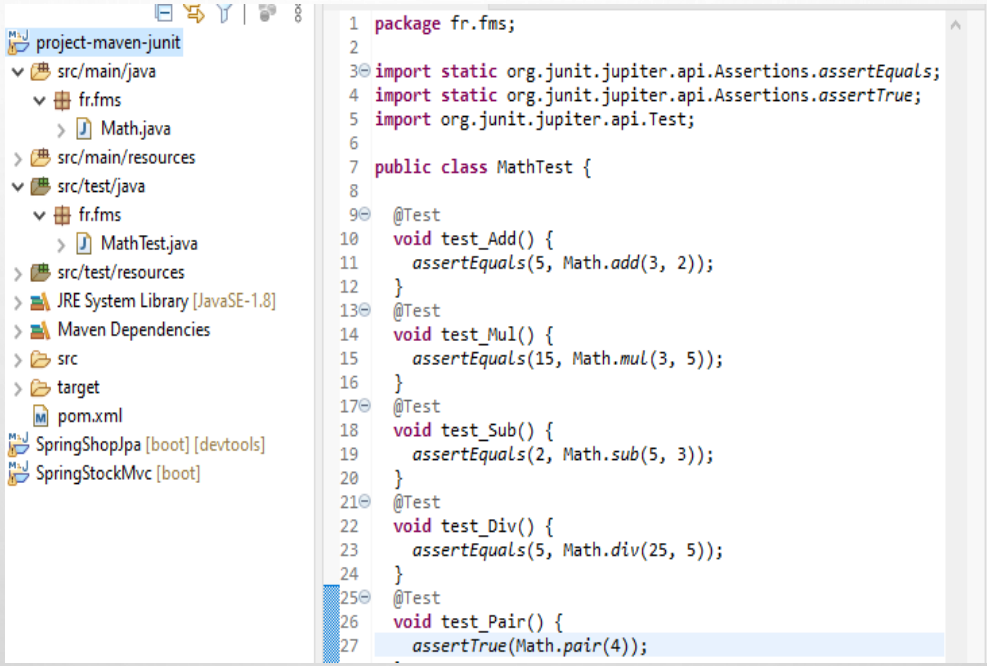
Plus un bug est détecté tôt, moins son coût de correction est élevé :

- Moins on réalise de tests unitaires, plus on va devoir faire de tests manuels pour le débusquer (mauvaise pratique)
- Plus notre couverture de tests unitaires est de qualité et importante, plus vite un bug est détecté (bonne pratique)

MAVEN ET JUNIT



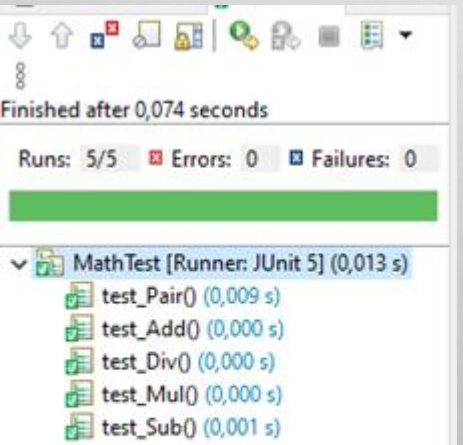
Ajouter les dépendances Junit dans pom.xml
Puis ajouter une classe à tester et une classe de test dans les répertoires




```java
package fr.fms;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertTrue;
import org.junit.jupiter.api.Test;

public class MathTest {

  @Test
  void test_Add() {
    assertEquals(5, Math.add(3, 2));
  }
  @Test
  void test_Mul() {
    assertEquals(15, Math.mul(3, 5));
  }
  @Test
  void test_Sub() {
    assertEquals(2, Math.sub(5, 3));
  }
  @Test
  void test_Div() {
    assertEquals(5, Math.div(25, 5));
  }
  @Test
  void test_Pair() {
    assertTrue(Math.pair(4));
```

```java
package fr.fms;

public class Math {

      public static int add(int num1, int num2)
        return num1 + num2;
      }

      public static int mul(int num1, int num2)
        return num1 * num2;
      }

      public static int sub(int num1, int num2)
          return num1 - num2;
      }

      public static int div(int num, int by) {
        return num/by;
      }

      public static boolean pair(int num) {
          return num%2 == 0;
      }
}
```

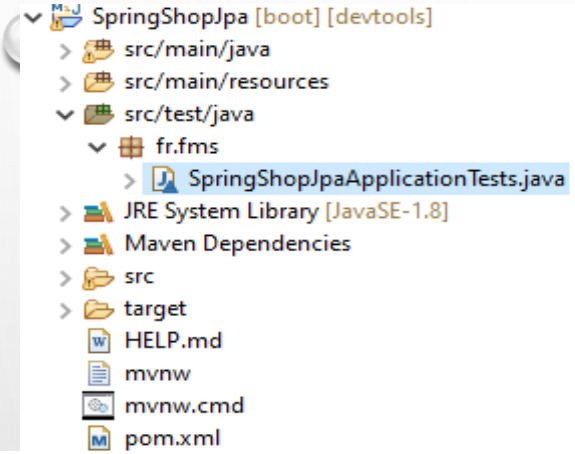
Pour exécuter les tests :
Project/clic droit/run as / junit test



C:\... mvn test

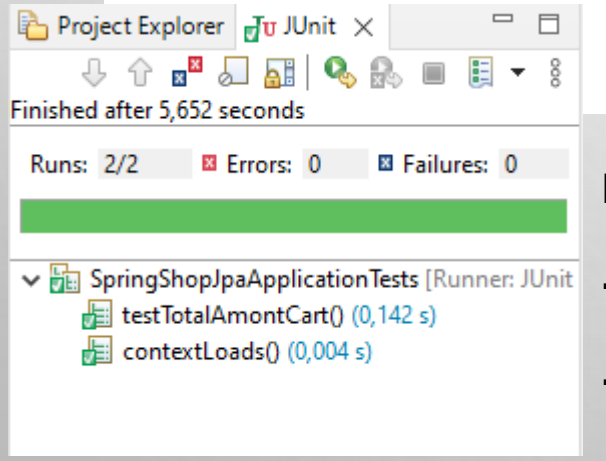
SPRING BOOT, MAVEN ET JUNIT

On souhaite vérifier si le montant total de notre panier est cohérent avec les prix des articles ajoutés qq soit les quantités.




```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

```java
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;

import fr.fms.business.IBusinessImpl;
import fr.fms.entities.Article;

@SpringBootTest
class SpringShopJpaApplicationTests {
    @Autowired
    IBusinessImpl business;

    @Test
    void contextLoads() {
        assertFalse(1==2);
    }

    @Test
    void testTotalAmontCart() {
        business.addArtToCart(new Article((long)1,"Samsung","Samsung S8",250,1,null));
        business.addArtToCart(new Article((long)2,"Samsung","Samsung S9",250,1,null));
        business.addArtToCart(new Article((long)3,"IPhone","IPhone 10",500,1,null));
        business.addArtToCart(new Article((long)1,"Samsung","Samsung S8",250,1,null));

        assertEquals(business.getTotalAmount(),1250);
    }
}
```



Pour 2 tests lancés avec succès :

- **contextLoads** vérifier que l'assertion est bien fausse

- **testTotalAmountCart** vérifie que le montant du panier est cohérent aux ajouts

Failures = nb de test en échec
Errors = nb d'erreur de compil

Project Explorer | JUnit

Finished after 9,071 seconds

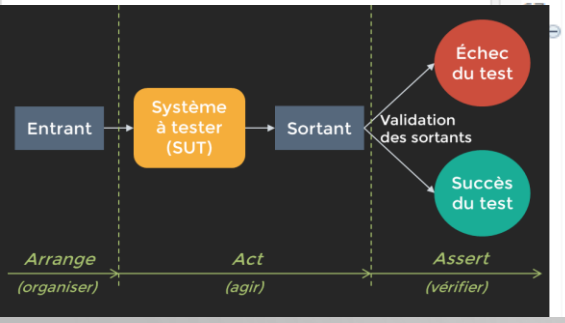
Runs: 7/7 Errors: 0 Failures: 0

- SpringShopJpaApplicationTests [Runner: JUnit
  - multiply_shouldReturnZero(int) (0,176 s)
    - 1 x 0 doit être égal à 0 (0,176 s)
    - 2 x 0 doit être égal à 0 (0,008 s)
    - 42 x 0 doit être égal à 0 (0,005 s)
    - 1011 x 0 doit être égal à 0 (0,003 s)
    - 5089 x 0 doit être égal à 0 (0,004 s)
  - orderShouldComputeLess1Second() (0,511
  - testTotalAmontCart() (0,006 s)

Failure Trace

Console | SpringShopJpaApplicationTests.java | SpringShopJpa/pom.xml | ArticleRepository.java

```java
package fr.fms;

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

@SpringBootTest
class SpringShopJpaApplicationTests {
    @Autowired
    IBusinessImpl business;

    private static Instant startedAt;

    @BeforeEach
    public void beforeEachTest() {
        System.out.println("avant chaque test");
    }

    @AfterEach
    public void afterEachTest() {
        System.out.println("après chaque test");
    }

    @BeforeAll
    public static void initStartingTime() {
        System.out.println("Appel avant tous les tests");
        startedAt = Instant.now();
    }

    @AfterAll
    public static void showTestDuration() {
        System.out.println("Appel après tous les tests");
        final Instant endedAt = Instant.now();
        final long duration = Duration.between(startedAt, endedAt).toMillis();
        System.out.println(MessageFormat.format("Durée des tests : {0} ms", duration));
    }

    @ParameterizedTest(name = "{0} x 0 doit être égal à 0")
    @ValueSource(ints = { 1, 2, 42, 1011, 5089 })
    public void multiply_shouldReturnZero(int arg) {
        assertEquals(0, arg*0);
    }

    @Timeout(1)
    @Test
    public void orderShouldComputeLess1Second() {
        business.order();
    }

    @Test
    void testTotalAmontCart() {
        // Arrange
        business.addArtToCart(new Article((long)1,"Samsung","Samsung S8",250,1,null));
        business.addArtToCart(new Article((long)2,"Samsung","Samsung S9",250,1,null));
        business.addArtToCart(new Article((long)3,"IPhone","IPhone 10",500,1,null));

        // Act
        double amount = business.getTotalAmount();

        // Assert
        assertEquals(amount,1000);
    }
```




```
Appel avant tous les tests
13:51:18.517 [main] DEBUG org.springframewo
13:51:18.522 [main] DEBUG org.springframewo

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.6.0)

2022-06-26 13:51:18.878  INFO 16604 --- [
2022-06-26 13:51:18.879  INFO 16604 --- [
2022-06-26 13:51:20.017  INFO 16604 --- [
2022-06-26 13:51:20.114  INFO 16604 --- [
2022-06-26 13:51:21.026  INFO 16604 --- [
2022-06-26 13:51:21.122  INFO 16604 --- [
2022-06-26 13:51:21.396  INFO 16604 --- [
2022-06-26 13:51:21.665  INFO 16604 --- [
2022-06-26 13:51:21.759  INFO 16604 --- [
2022-06-26 13:51:21.780  INFO 16604 --- [
2022-06-26 13:51:22.855  INFO 16604 --- [
2022-06-26 13:51:22.866  INFO 16604 --- [
2022-06-26 13:51:23.859  WARN 16604 --- [
2022-06-26 13:51:24.403  INFO 16604 --- [
2022-06-26 13:51:25.981  INFO 16604 --- [

Using generated security password: be568f54

2022-06-26 13:51:26.007  INFO 16604 --- [
avant chaque test
après chaque test
avant chaque test
après chaque test
avant chaque test
après chaque test
avant chaque test
après chaque test
avant chaque test
après chaque test
avant chaque test
après chaque test
avant chaque test
après chaque test
avant chaque test
après chaque test
Appel après tous les tests
Durée des tests : 8 362 ms
```

TDD : Test Driven Development
Ou
Développement piloté par les tests

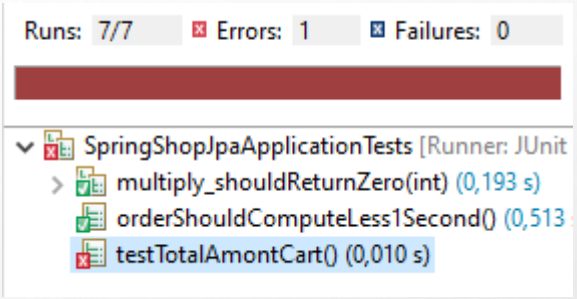
En règle générale, on a tendance à coder une fonctionnalité puis réaliser le test associé, dans cette approche, on code le test en fonction du besoin, d'abord le test sera en échec

```
68    @Test
69    void testTotalAmontCart() {
70        // Arrange
71        business.addArtToCart(new Article((long)1,"Samsung","Samsung S8",250,1,null));
72        business.addArtToCart(new Article((long)2,"Samsung","Samsung S9",250,1,null));
73        business.addArtToCart(new Article((long)3,"IPhone","IPhone 10",500,1,null));
74
75        // Act
76        double amount = business.getTotalAmount();
77
78        // Assert
79        assertEquals(amount,1000);
80    }
```

The method getTotalAmount() is undefined for the type IBusinessImpl

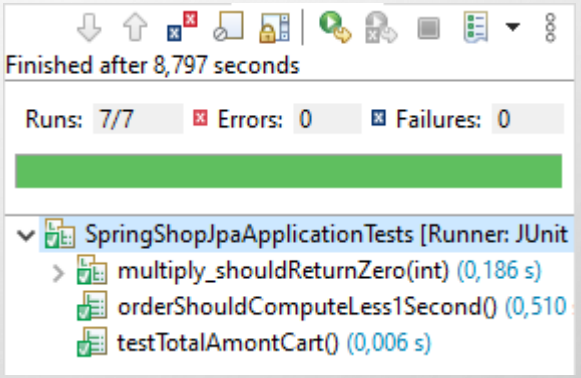
2 quick fixes available:

Create method 'getTotalAmount()' in type 'IBusinessImpl'



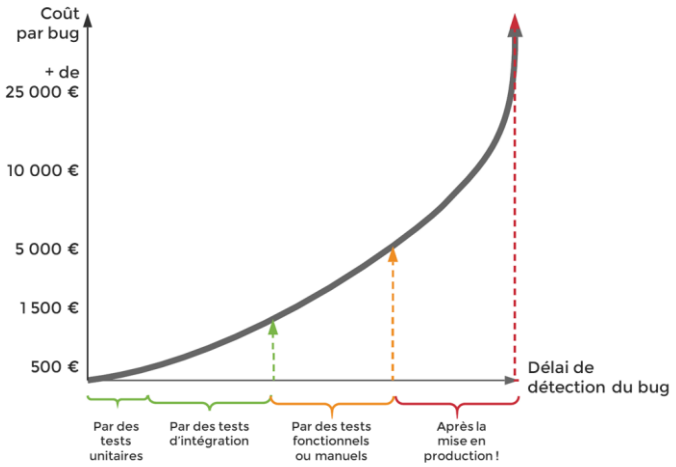
Puis on ajoute le code dans la méthode de la classe :

```
public double getTotalAmount() {
    double total = 0;
    for(Article article : cart.values()) {
        total += article.getPrice()*article.getQuantity();
    }
    return total;
}
```



Cette approche garantie qu'on reste centré sur les fonctionnalités
De plus, le code écrit après le test est plus simple à tester,
plus clair à comprendre et plus facile à maintenir : Robuste

ASSERTJ



La courbe montre que + un bug est découvert tardivement, plus il est couteux de le corriger aussi comment écrire des tests :

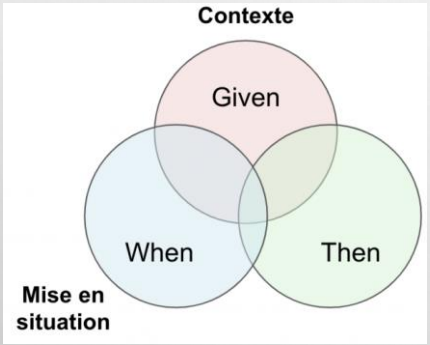
- Simples (à tester)
- Clairs (à comprendre)
- Faciles (à maintenir)

→ AssertJ apporte à Junit tout ce complément

Cas de test	**Assertions JUnit**	**Assertions AssertJ**
Un nom est compris entre 5 et 10 caractères.	assertTrue(name.length > 4 && name.length < 11);	assertThat(name) .hasSizeGreaterThan(4) .hasSizeLessThan(11);
Un nom est situé dans la première moitié de l'alphabet	assertTrue(name.compareTo("A") >= 0 && name.compareTo("M") <= 0);	assertThat(name).isBetween("A", "M");
Une date et heure locale se situent aujourd'hui ou dans le futur.	assertTrue(dateTime.toLocalDate().isAfter(LocalDate.now()) || dateTime.toLocalDate().isEqual(LocalDate.now()));	assertThat(dateTime.toLocalDate()) .isAfterOrEqualTo(LocalDate.now());

@DataJpaTest

Si vous souhaitez maintenant tester vos entités Jpa dans une base de données virtuelle avec l'accès aux données persistantes sans modifications possibles :




```java
import org.junit.jupiter.api.Test;

@RunWith(SpringRunner.class)
@DataJpaTest
@AutoConfigureTestDatabase(replace=Replace.NONE)
public class SpringShopJpaTests {
    @Autowired
    ArticleRepository articleRepository;

    @Autowired
    CategoryRepository categoryRepository;

    @Test
    void test_add_article() {
        //GIVEN
        Category anonymous = categoryRepository.save(new Category(null,"anonymous",null));
        articleRepository.save(new Article(null,"incognito","incognito 007" , 375 , 1 , anonymous));

        //WHEN
        Article article = articleRepository.findByBrandContains("incognito").get(0);

        //THEN
        assertEquals("incognito 007", article.getDescription());
    }

    @Test
    void should_find_one_article() {
        Iterable<Article> articles = articleRepository.findAll();
        assertThat(articles).isNotEmpty();
    }
}
```

Notre base n'a subit aucun changement

```
| 21 | Panasonic | HT           |  1500 |        1 |          6 |
| 22 | Philips   | L43          |   450 |        1 |          6 |
| 26 | samsung   | samsung S200 |  1500 |        1 |          1 |
| 29 | Iphone    | Iphone 10    |   500 |        1 |          1 |
+----+-----------+--------------+-------+----------+------------+
24 rows in set (0.000 sec)

MariaDB [Stock]>
```

Tester un Controller
(MockMvc & Mockito)

Debug | Project Explorer | JUnit

Finished after 7,661 seconds

Runs: 1/1 | Errors: 0 | Failures: 0

SpringShopControllersTests [Runner: JUnit 5] (0,239 s)

test_get_welcome() (0,239 s)

SpringShopControllersTests.java

```
1  package fr.fms;
2
3  import static org.hamcrest.CoreMatchers.containsString;
22
23 @SpringBootTest
24 @AutoConfigureMockMvc
25 public class SpringShopControllersTests {
26
27     @Autowired
28     private MockMvc mvc;
29
30     @Test
31     public void test_get_welcome() throws Exception {
32         //GIVEN
33         mvc.perform(get("/"))
34             .andExpect(status().isOk())
35             .andExpect(content().string(containsString("Hello")));
36     }
37 }
```

```
@RequestMapping("/")
public @ResponseBody String home() {
    return "Hello";
}
```

```
import static org.mockito.Mockito.when;
3  import static org.hamcrest.CoreMatchers.containsString;
24
25 @WebMvcTest(ArticleController.class)
26 public class SpringShopControllersTests {
27
28     @Autowired
29     private MockMvc mvc;
30
31     @MockBean
32     private IBusinessImpl businessImpl;
33
34     @Test
35     public void test_get_welcome() throws Exception {
36         when(businessImpl.great()).thenReturn("Hello, Mock");
37
38         this.mvc.perform(get("/greating"))
39             .andDo(print())
40             .andExpect(status().isOk())
41             .andExpect(content()
42             .string(containsString("Hello, Mock")));
43     }
44 }
```

Runs: 1/1 | Errors: 0 | Failures: 0

SpringShopControllersTests [Runner: JUnit 5] (0,169 s)

test_get_welcome() (0,169 s)

On souhaite ici renvoyer une « Hello, Mock » à la place de « Hello World » et vérifier si c'est bon

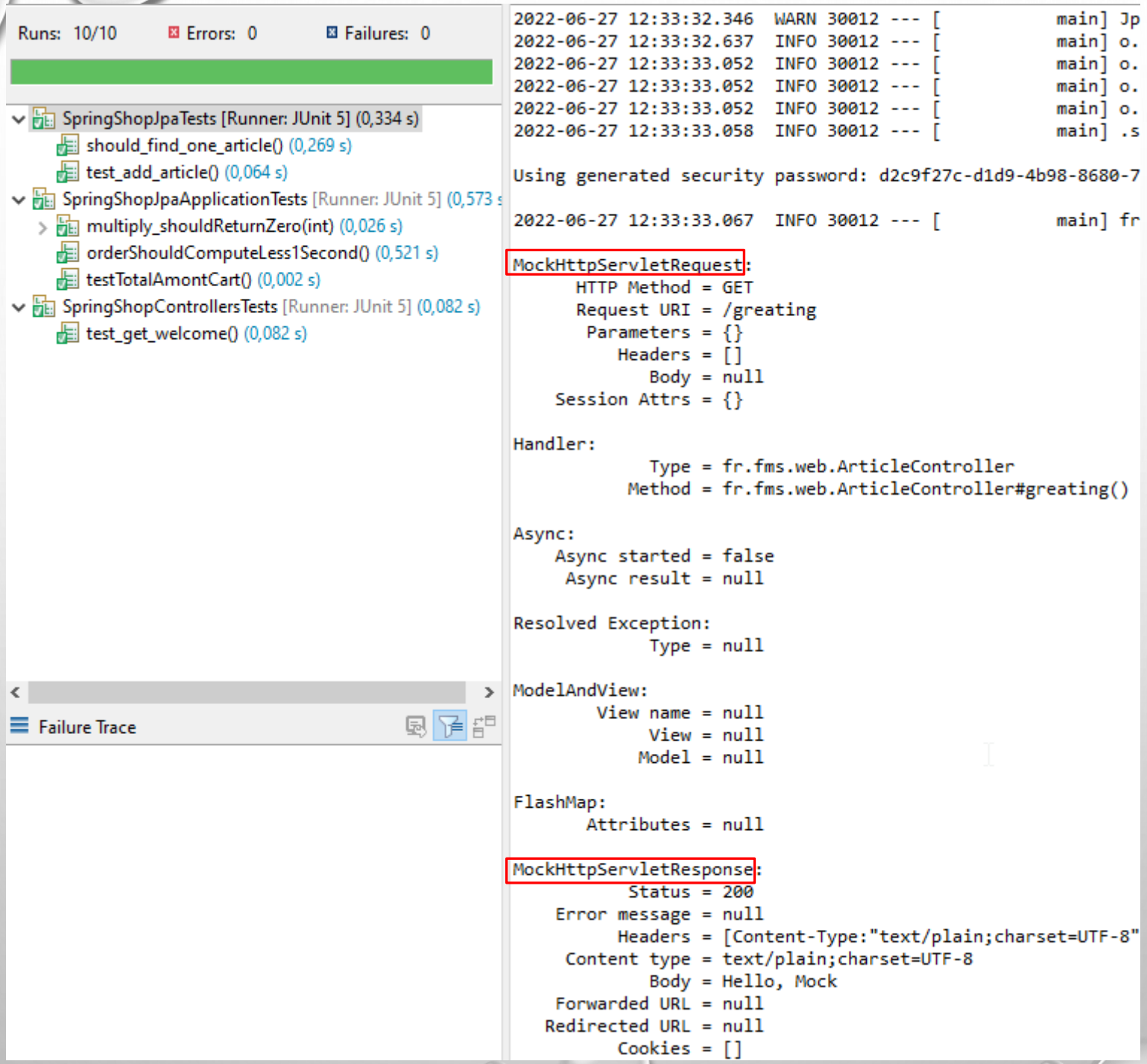
```
public String great() {
    return "Hello World";
}
```

```
@RequestMapping("/greating")
public @ResponseBody String greating() {
    return businessImpl.great();
}
```

Voir résultat slide suivante

Ex avec une api, à suivre

```
@Test
public void test_get_articles() throws Exception {
    //.................
    this.mvc.perform(get("/articles"))  //THEN
        .andExpect(status().isOk())
        .andExpect(jsonPath("$[0].brand", is("Samsung")));
        //$    : pointe sur la racine de la structure JSON
        //[0]  : pour le 1er élément
        //brand: brand pour l'attribut
        //is   : correspond au résultat attendu
}
```



MAVEN

- C'est un outil qui permet l'automatisation du processus de développement d'un projet java, il nous sera utile tout au long du cycle de vie de nos applications.
- Exemples :

  → **mvn compile** est une commande demandant à maven de compiler l'ensemble du code source de votre projet

  → **Mvn package** va générer un jar(appli desktop notamment) ou war(web app)

  → Mvn test/install/deploy/site…

- Un projet maven contient un fichier pom.Xml(project object model) notamment pour gérer l'injection des dépendances via un repository local ou distant :

  → c:\users\moham\.M2\repository

- *Nb : Spring peut utiliser maven ou gradle*

Déploiement avec maven

Sous Linux, il faut d'abord installer maven :
→ sudo apt-get update + sudo apt install maven
→ mvn package : compile + test + génére le .jar dans target

Sous Windows, fichier pom.xml/clic droit/ run as / maven build…

Dans le dossier Target donc vous pouvez récupérer « SpringShopMvc-0.0.1-SNAPSHOT.jar » puis le copier coller sur votre bureau et fermer Eclipse avant de l'exécuter.

Puis rendez vous sur un navigateur pour utiliser l'application normalement

Il est possible sur un même réseaux local de faire un appel depuis un poste client vers un poste dit serveur contenant l'appli qui tourne. Pour ce faire, à la place de localhost, insérer l'adresse IP du poste serveur.

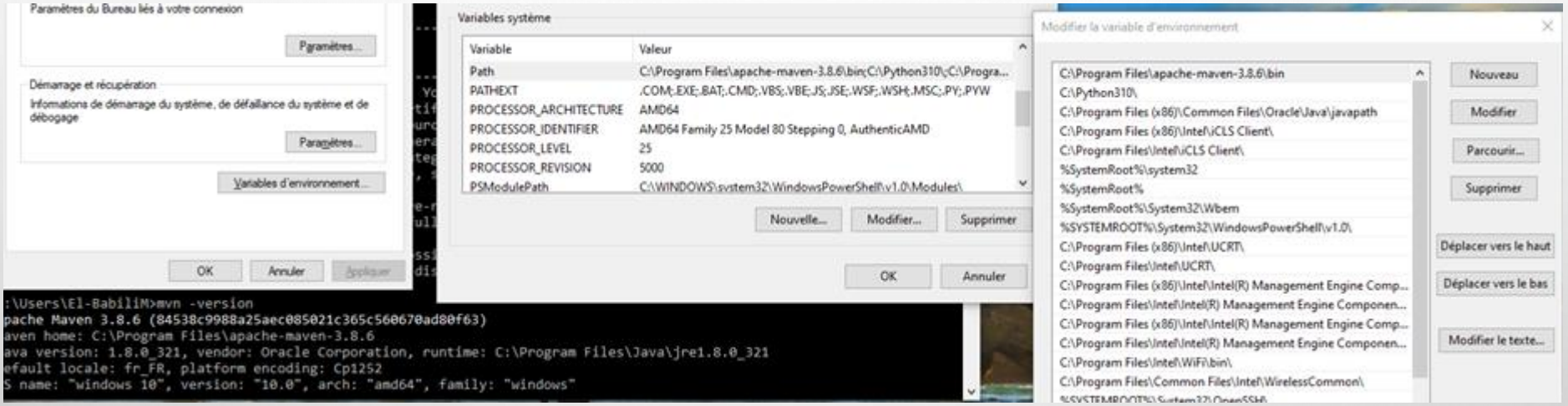
Commande pour obtenir votre Ip sous Linux : hostname –i

Sous windows : ipconfig

NB : marche pas avec les pc airbus hyper verrouillé ;)

Installer Maven pour l'utiliser en ligne de commande

Télécharger sur le site le zip « apache-maven-3.8.6-bin.zip » le dézipper dans programmes
Puis ajouter à la variables d'environnement le chemin vers le rep bin
Tester en mode console si c'est ok
ToDo : Voir et préparer suite ici



A RETENIR

Rappelons que les tests unitaires ont pour vocation à tester uniquement le contenu d'une méthode, alors que les tests d'intégration impliquent de tester plus largement une fonctionnalité.

La génération de notre Jar est précédé par l'ensemble des tests :

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 10, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- maven-jar-plugin:3.2.0:jar (default-jar) @ SpringShopJpa ---
[INFO] Building jar: C:\Users\El-BabiliM\SpringWorkspace\SpringShopJpa\target\SpringShopJpa-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.6.0:repackage (repackage) @ SpringShopJpa ---
[INFO] Replacing main artifact with repackaged archive
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  11.607 s
[INFO] Finished at: 2022-06-27T14:35:26+02:00
[INFO] ------------------------------------------------------------------------
```

Voir aussi :
https://www.baeldung.com/mockito-series
https://cucumber.io/
Spring Boot Unit and Intégration Testing Overview

ETAPES SUIVANTES

- Vérifier la couverture de nos tests avec Jacoco par ex
- Améliorer la qualité du code avec le formatage et la détection de code mal écrit (Outils Sonar)
- Sécuriser notre appli en vérifiant les dépendances (OWASP Dependency Check)

→ Maven va nous aider à automatiser ces outils durant le build

→ SonarQube va nous aider à observer tous ces indicateurs comme dans un cockpit

→ Cucumber permet d'écrire des scénarios de tests(BDD/US) repris sous forme de steps (Given, When, Then)

→ Jenkins va gérer le tout à l'aide de pipeline :