

TP : Intégration Continue

Part 1 : Tests unitaires

Objectif :

Introduction à l'intégration continue (tests unitaires avec l'IDE IntelliJ).



Prérequis :

- Installation de Maven (Télécharger [ici](#) et suivez les instructions [suivantes](#) pour installer).
- Installation d'un IDE JAVA (l'ensemble des captures faites sont avec IntelliJ).

Protocole :

Le but de ce TP est d'appliquer l'intégration continue aux projets JAVA. Et par la suite la manipulation des différents outils et procédures nécessaires aux tests unitaires.

Exercice I : Calcul Géométrique

Création d'un projet **JAVA** de calculs arithmétiques et géométriques. Et définition des classes permettant de lancer les tests unitaires avec l'IDE **IntelliJ**.

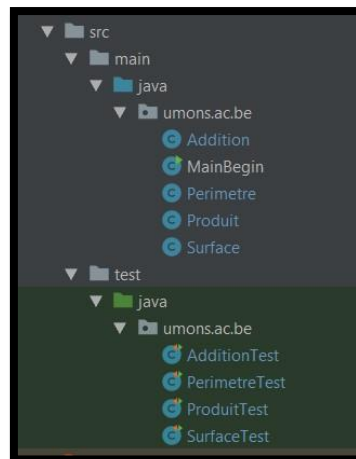


Figure 1 : Le package du projet « *Calculs_Geo* »

Exercice II : Gestion de monnaie et devise

Création d'un projet **JAVA** de transfert et manipulation de Monnaie. Et lancement des tests unitaires avec l'IDE **IntelliJ** et via **Maven**.

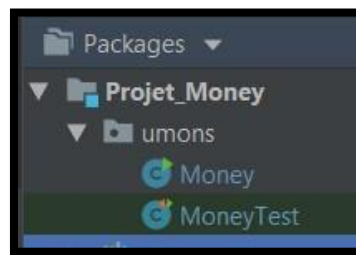


Figure 2 : Le package du projet « *Money* »

Exercice III : Gestion des profs et cours

Définition et lancement des tests unitaires avec **IntelliJ** et **Maven** pour un projet **JAVA** de gestion des profs et de cours.

Exercice I : Périmètre et Surface

Création des classes java et définition des classes de test.

Lancement des tests unitaires avec l'IDE IntelliJ.

Enoncé :

Il s'agit de définir des méthodes de classes pour faire des calculs arithmétiques tels que l'addition et la multiplication, permettant de réaliser des calculs géométriques tels que la surface et le périmètre. Ensuite de lancer des tests unitaires en utilisant des classes de tests pour chaque classe.

Questions :

Question 1 : Créez un projet **JAVA** sous le nom « **Calculs_Geo** ».

Question 2 : Ecrire la classe **Addition** qui implémente l'en-tête ci-dessous.

```
package umons.ac.be;

public class Addition {

    public static int add(int a, int b) {
        return a + b;
    }
}
```

Figure 3 : La classe **Addition**

Question 3 : Définir une classe « **Main** » contenant la fonction **main(...)** du programme. Appelez la méthode **add** de la classe **Addition**. Run ensuite le **main()**

Question 4 : Créer la classe **AdditionTest** qui teste la classe **Addition** La création d'une classe test se fait en sélectionnant la classe à tester puis en sélectionnant **Navigate -> test** ou **CTRL+Maj+T**. **Remarque :** veillez à ce que la classe de test soit **public**

```
package umons.ac.be;

import ...

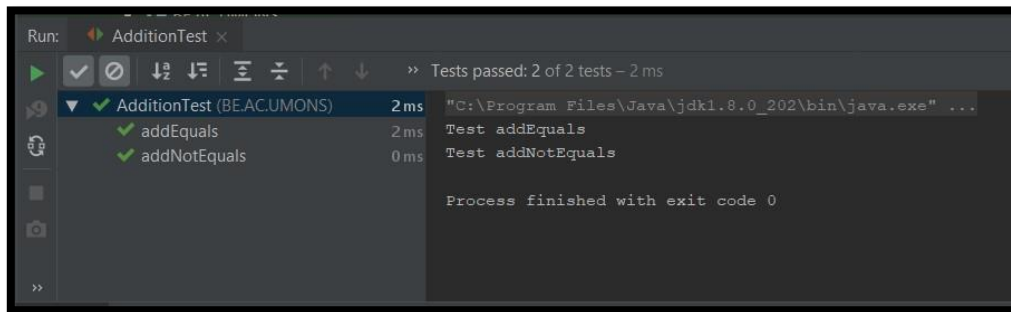
public class AdditionTest{

    @Test
    public void addEquals() {
        System.out.println("Test addEquals");
        assertEquals(Addition.add( 5, 3), 8);
    }

    @Test
    public void addNotEquals() {
        System.out.println("Test addNotEquals");
        assertEquals(Addition.add( 1, 1), 3);
    }
}
```

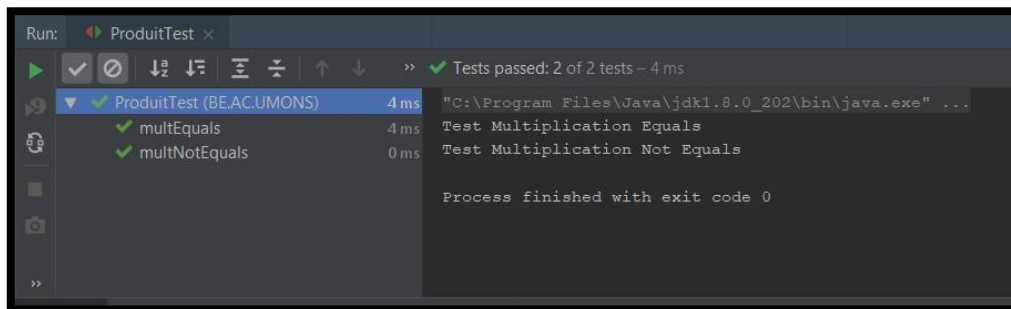
Figure 4 : La classe AdditionTest

Question 5 : Lancez les tests unitaires définis dans la classe de test AdditionTest sur l'IDE.

Figure 5 : Les résultats des tests unitaires de la classe Addition

Question 6 : Définir une classe Produit qui calcule le produit de deux entiers.

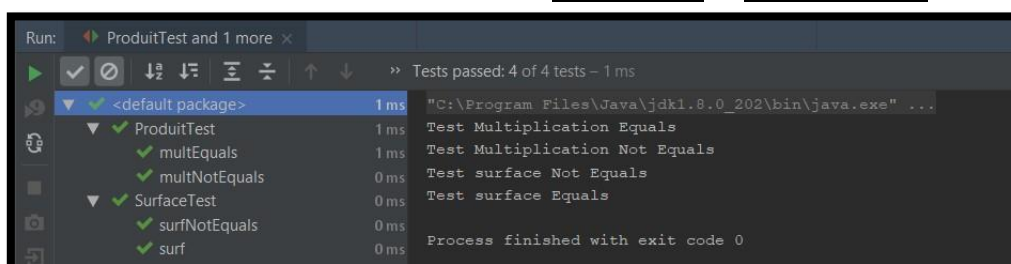
Question 7 : Définir une classe ProduitTest qui test la classe Produit et lancer les tests unitaires.

Figure 6 : Les résultats des tests unitaires de la classe Produit

Question 8 : Définir deux classes Surface ($a * b$) et Périmètre ($(a + b) * c$) qui font appel aux fonctions add() et mult() des classes Addition et Produit.

Question 9 : Définir les classes SurfaceTest et PérimètreTest contenant les différents tests unitaires des classes Surface et Périmètre.

Question 10 : Lancez les tests unitaires des classes SurfaceTest et PérimètreTest avec l'IDE.

Figure 7 : Les résultats des tests unitaires des classes Surface et Périmètre

Exercice II : Gestion de monnaie et devise

Création du projet MAVEN.

Lancement des tests unitaires sur l'IDE IntelliJ et avec Maven.

Enoncé :

Il s'agit de définir une classe Money permettant de représenter un montant (mAmount) dans une certaine devise (mCurrency). Le montant doit être positif et la devise s'exprime selon la norme ISO (EUR € euro, USD \$, CHF Franc Suisse, GBP Livre sterling...). L'addition de deux monnaies, se fait pour des devises identiques.

Questions :

Question 1 : Vérifiez l'installation de Maven avec la commande :

mvn -version



```
Command Prompt
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

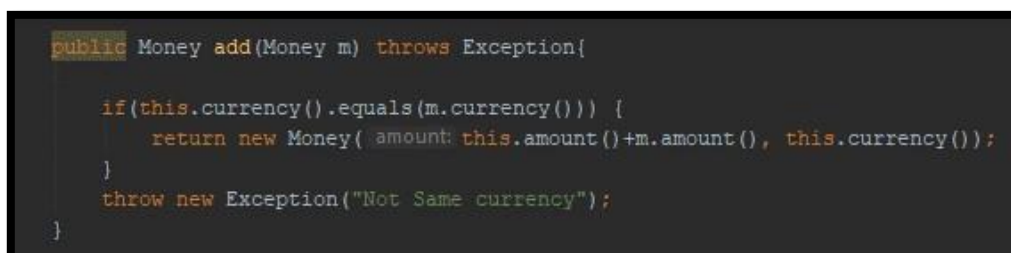
C:\Users\Yassine Amkrane>mvn -version
Apache Maven 3.6.2 (40f52333136460af0dc0d7232c0dc0bcf0d9e117; 2019-08-27T17:06:16+02:00)
Maven home: c:\maven362\bin\..
Java version: 1.8.0_202, vendor: Oracle Corporation, runtime: C:\Program Files\Java\jdk1.8.0_202\jre
Default locale: en_GB, platform encoding: Cp1252
OS name: "windows 10", version: "10.0", arch: "amd64", family: "windows"

C:\Users\Yassine Amkrane>
```

Figure 8 : La vérification De la version de Maven

Question 2 : Créez un projet **Maven** et récupérer le code non finalisé des deux classes « **Money** » et « **MoneyTest** ». (Le projet est disponible sur Moodle).

Question 3 : Définir la méthode de classe **add(Money m)** qui permet d'ajouter la monnaie.



```
public Money add(Money m) throws Exception{
    if(this.currency().equals(m.currency())) {
        return new Money( amount: this.amount()+m.amount(), this.currency());
    }
    throw new Exception("Not Same currency");
}
```

Figure 9 : La méthode de classe **add(Money m)**

Question 4 : S'inspirer de la méthode **testAmount()** de la classe **MoneyTest** pour écrire une méthode test **testCurrency()** qui teste la méthode **currency()**.

⑨ Utiliser la fonction **assertEquals()** au lieu de **assertNotEquals()**.

```
@Test
public void testCurrency(){
    System.out.println("Currency Test");
    Money instance = new Money( amount: 12, currency: "EUR");
    String expectedResult = "EUR";
    String result = instance.currency();
    assertEquals(expectedResult,result);
}
```

Figure 10 : La méthode **testAmount()** de la classe **MoneyTest**

Question 5 : S'inspirer de la méthode **testAdd Money()** pour définir une méthode de test **testAdd()** qui utilise la fonction **add(int, String)**.

```
@Test
public void testAdd() throws Exception {
    System.out.println("Add Test");
    int namount = 10;
    String currency = "EUR";
    Money instance = new Money( amount: 180, currency: "EUR");
    Money expectedResult = new Money( amount: 190, currency: "EUR");
    Money result = instance.add(namount,currency);
    assertTrue( message: "There are Equals", condition: expectedResult.amount()==result.amount() && expectedResult.currency().equals(result.currency()));
    // TODO review the generated test code and remove the default call to fail.
    //fail("The test case is a prototype.");
}
```

Figure 11 : La méthode de test **testAdd()**

Question 6 : Lancer les tests unitaires de la classe **MoneyTest**.

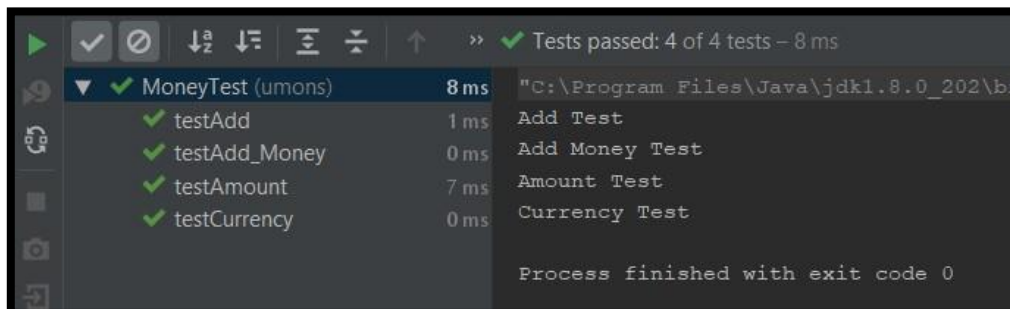


Figure 12 : Les résultats des tests unitaires de la classe **MoneyTest**

Question 7 : Sur le terminal de l'IDE, lancez les tests avec **Maven** (Commande : **mvn test**).

```

Terminal: Local x +
tinue\2020-2021\TP0\Projets\TP0 - Projets_Solution\Money\src\main\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:compile (default-compile) @ Money ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:3.0.2:testResources (default-testResources) @ Money ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory D:\X-Documents\Documents\TPs - Cours - Projets\TPs Intégration Continue\2020-2021\TP0\Projets\TP0 - Projets_Solution\Money\src\test\resources
[INFO]
[INFO] --- maven-compiler-plugin:3.8.0:testCompile (default-testCompile) @ Money ---
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ Money ---
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running be.ac.umons.MoneyTest
Add Test
Add Money Test
Amount Test
Currency Test
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.021 s - in be.ac.umons.MoneyTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 4, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.952 s
[INFO] Finished at: 2020-10-21T16:35:39+02:00
[INFO] -----
D:\X-Documents\Documents\TPs - Cours - Projets\TPs Intégration Continue\2020-2021\TP0\Projets\TP0 - Projets_Solution\Money>

```

Figure 13 : Le lancement des tests unitaires avec Maven

Exercice III : Gestion des profs et cours

Enoncé :

Il s'agit d'un projet JAVA de gestion de bases de données de professeurs et de cours. Le projet définit une classe qui permet la connexion à la base de données, et aussi des classes de gestion des Profs et de cours (Ajout, Modification, Suppression et interrogation...).

Questions :

Question 1 : Récupérer le projet Java « **TP_JAVA_Maven-master** » non finalisé. (voir sur Moodle)

Question 2 : Tester la connexion à la base (la classe **DataAccessTest**).

Remarque : N'oubliez pas d'importer la base de donnée (db.sql) dans phpmyadmin et de modifier le mot de passe par défaut (DBPASS) dans DataAccess.java avec votre mot de passe.

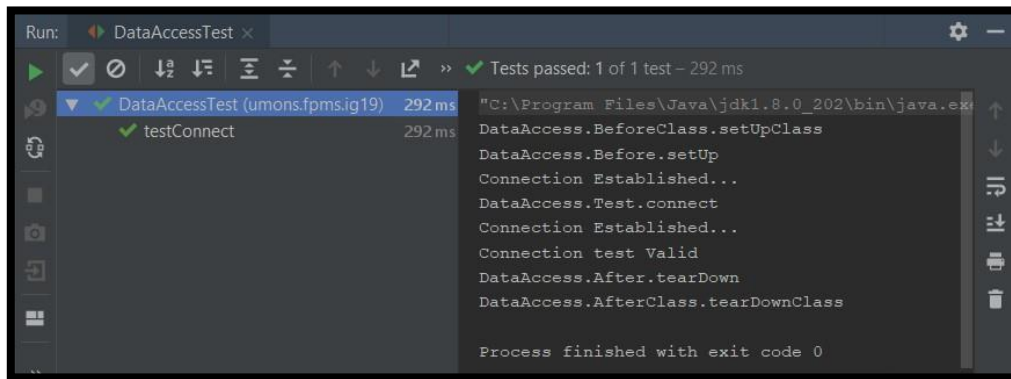


Figure 14 : Les résultats des tests unitaires de la classe *DataAccessTest*

Question 3 : Dans la classe **Cours**, s'inspirer de la méthode **public void add()** qui permet d'ajouter un cours pour écrire les méthodes **public void update(int id)** et **public void delete(int id)** qui permettent successivement de modifier et de supprimer des cours dans la base.

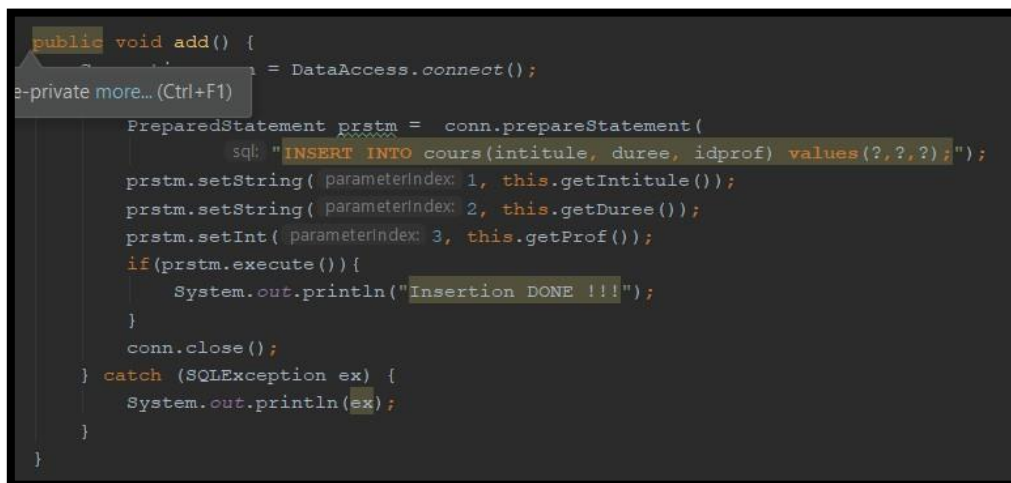


Figure 15 : La méthode **public void add()** de la classe *Cours*

Question 4 : Dans la classe **Prof**, écrire la méthode **public static ArrayList<Prof> select(String name)** qui permet de sélectionner les profs selon leur nom.

- ⑨ S'inspirer des méthodes **public static ArrayList<Prof> select()** et **public static Prof select(int id)** de la classe *Cours*


```

public static ArrayList<Prof> select() {
    Connection conn = DataAccess.connect();
    Statement stm;
    String sql = "SELECT * FROM prof;";
    ArrayList<Prof> result = new ArrayList<Prof>();
    try {
        stm = conn.createStatement();
        try (ResultSet rs = stm.executeQuery(sql)) {
            while(rs.next()) {
                Prof prof = new Prof(rs.getString(columnIndex: 2),
                                     rs.getString(columnIndex: 3),
                                     rs.getString(columnIndex: 4),
                                     rs.getString(columnIndex: 5));

                result.add(prof);
            }
            stm.close();
        }
        conn.close();
    } catch (SQLException ex) {
        System.out.println(ex);
    }
    return result;
}

```

Figure 16 : La méthode `public static ArrayList<Prof> select()` de la classe `Prof`

Question 5 : Dans la classe test `ProfTest`, s'inspirer de la méthode `testSelect_int()` pour définir la méthode de test `testSelect_String()` qui permet la sélection selon le nom d'un prof.

```

@Test
@Order(12)
public void testSelect_int() {
    System.out.println("Prof.Test.select_id");
    int id = 1;
    System.out.println(Prof.select(id).toString());
}

```

Figure 17 : La méthode de test `testSelect_int()` de la classe `ProfTest`

Question 6 : Dans la classe test `ProfTest`, s'inspirer de la méthode `testAdd()` pour définir les méthodes de test `testUpdate()` et `testDelete()`.

```

@Test
@Order(9)
public void testAdd() {
    System.out.println("Prof.Test.add");
    Prof p = new Prof("TJM_PROF0", "TJM_Prof0", "", ""); // The both last can be NULL.
    Prof instance = new Prof("TJM_PROF1", "TJM_Prof1", "10/05/1984", "Louvain");

    prof.add(); // This is the first insertion. So id == 1
    instance.add();
    p.add();

    instance = new Prof("TJM_PROF2", "TJM_Prof2", "10/10/1984", "Charleroi");
    instance.add();

    instance = new Prof("TJM_PROF3", "TJM_Prof3", "10/10/1987", "Paris");
    instance.add();

    System.out.println(Prof.select().toString());
}

```

Figure 18 : La méthode de test `testAdd()` de la classe `ProfTest` **Question**

7 : Lancez les tests unitaires de la classe **ProfTest**.

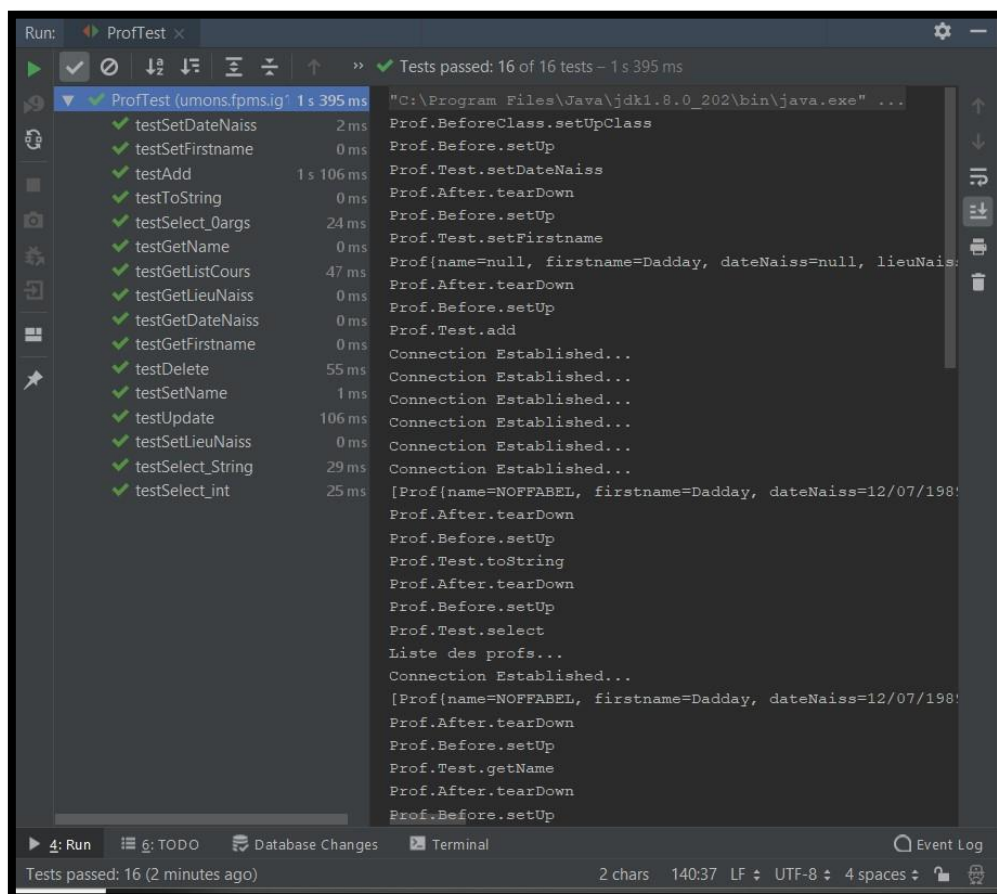


Figure 19 : Résultats des tests unitaires de la classe `ProfTest`.

Question 8 : De même, définir les méthodes de test **testUpdate()** et **testDelete()** dans la classe de

test **CoursTest**. Et lancer les tests unitaires pour cette classe.

```
/**
 * Test of add method, of class Cours.
 */
@Test
@Order(7)
public void testAdd() {
    System.out.println("Cours.Test.add");
    Cours instance = new Cours( intitule: "Java",  duree: "2h",  prof: 2);

    cours.add(); // id = 1

    instance.add(); // id = 2
    instance = new Cours( intitule: "IA",  duree: "3h",  prof: 1);
    instance.add(); // id = 3

    instance = new Cours( intitule: "C/C++",  duree: "3h",  prof: 2);
    instance.add(); // id = 4

    instance = new Cours( intitule: "Projet Encadrement",  duree: "3h",  prof: 3);
    instance.add(); // 5

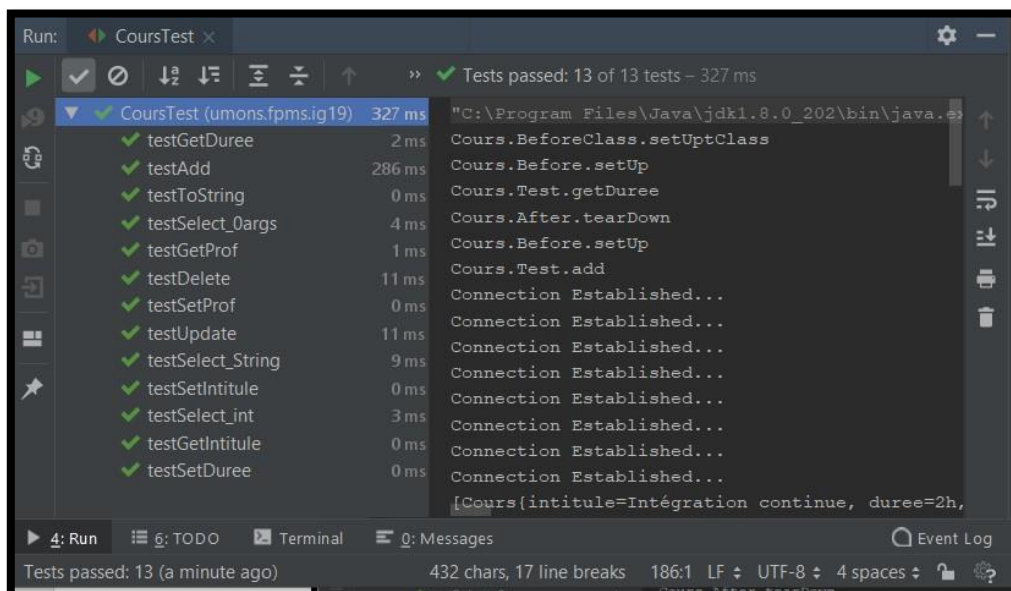
    instance = new Cours( intitule: "BDD",  duree: "3h",  prof: 1);
    instance.add(); // 6

    instance = new Cours( intitule: "Seminare",  duree: "4h",  prof: 3);
    instance.add(); // 7

    System.out.println(Cours.select().toString());
}
```

Figure 20 : La méthode de test **testAdd()** de la classe **Cours**

Question 9 : Lancez les tests unitaires de la classe **CoursTest**.



Question 10 : Sur le terminal de l'IDE, lancer les tests unitaires du projet avec **Maven**

⑨ Commande: `mvn clean test-Dtest=lg19Suite`

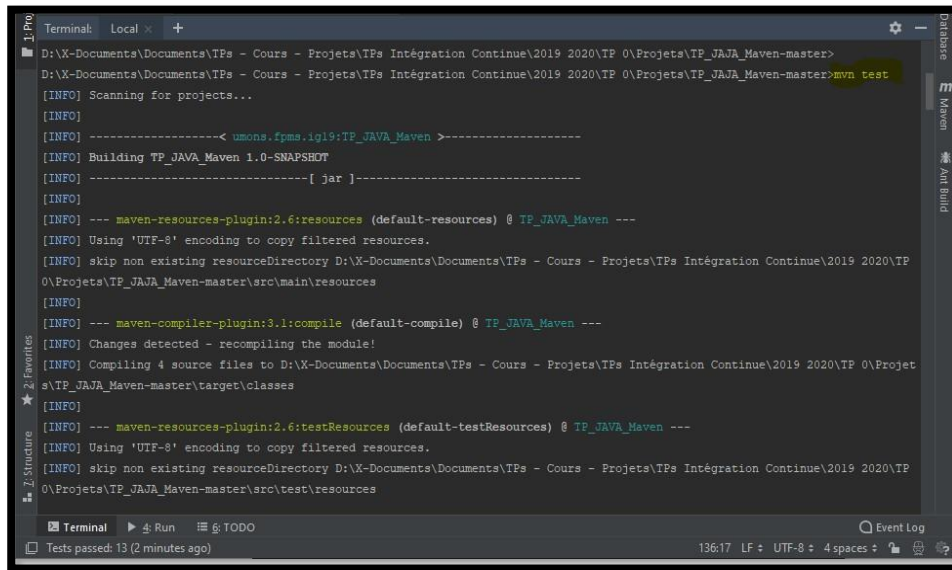


Figure 21 : Lancement des test unitaires avec Maven

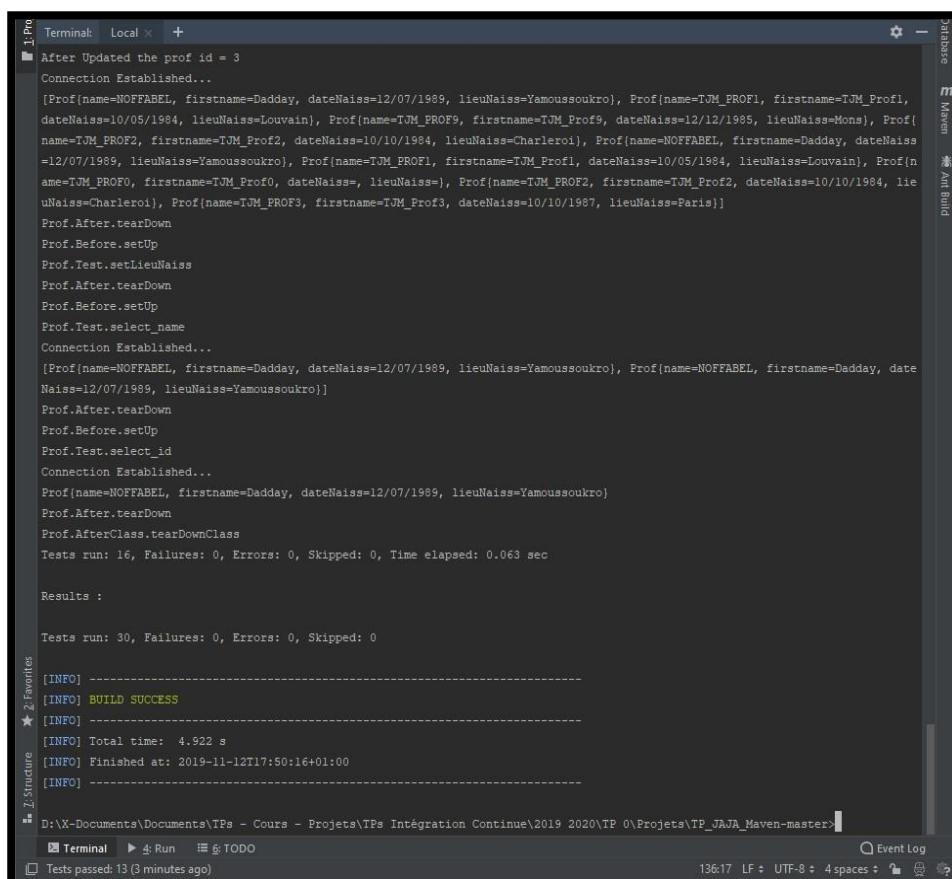


Figure 22 : Résultats des tests unitaires sur le Terminal

Question 11 : *<Optionnelle>* Au regard des logs, pouvons-nous optimiser ces tests ?

FIN de l'énoncé