Atelier Framework Spring-01
Maven-Partie-03

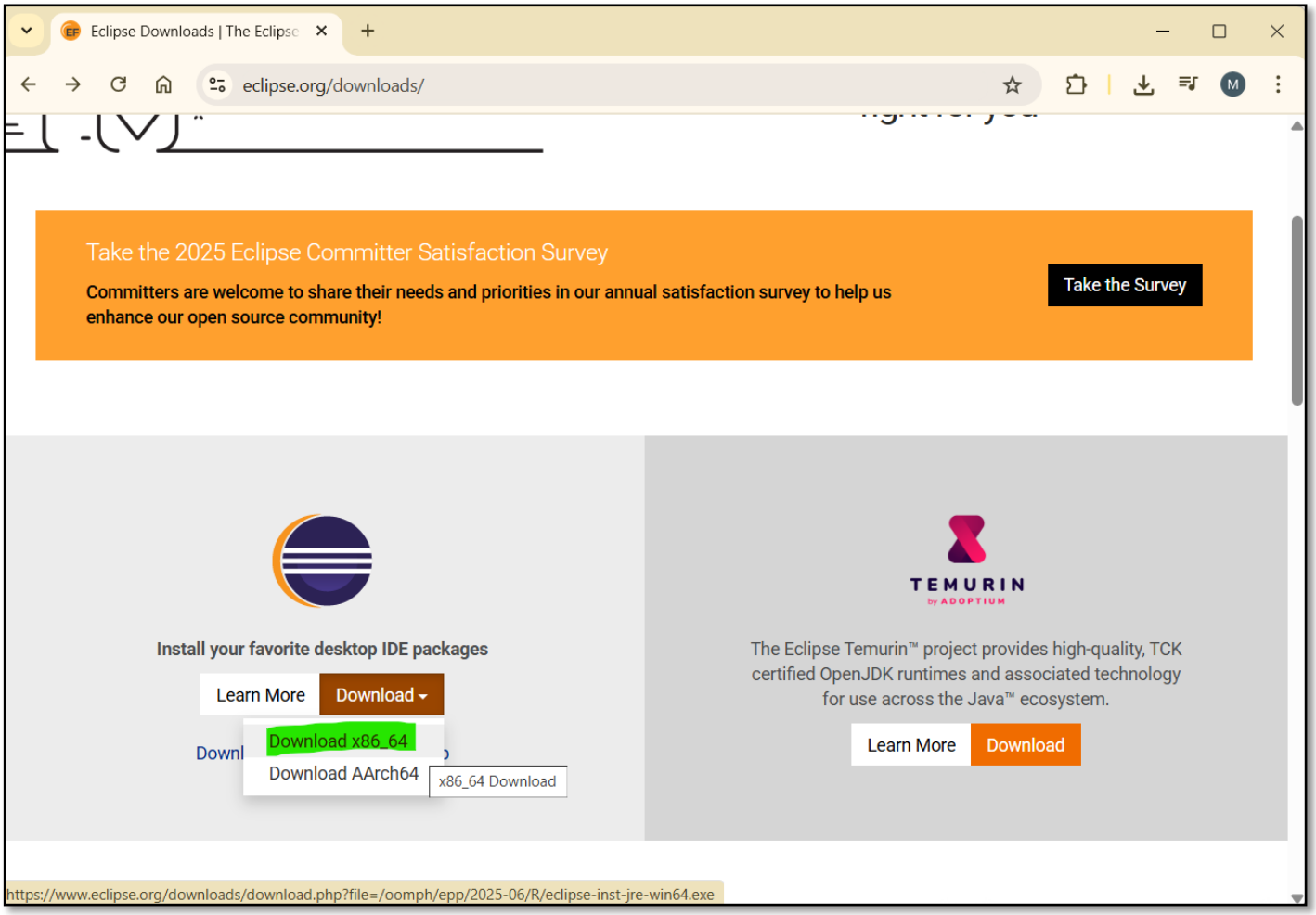
Objectifs

- **Utiliser des commandes Maven de base**
  - Importer un projet Maven à partir de « **eclipse** »
  - Passer des commandes Maven pour gérer le cycle de vie d'un projet « **Maven** »

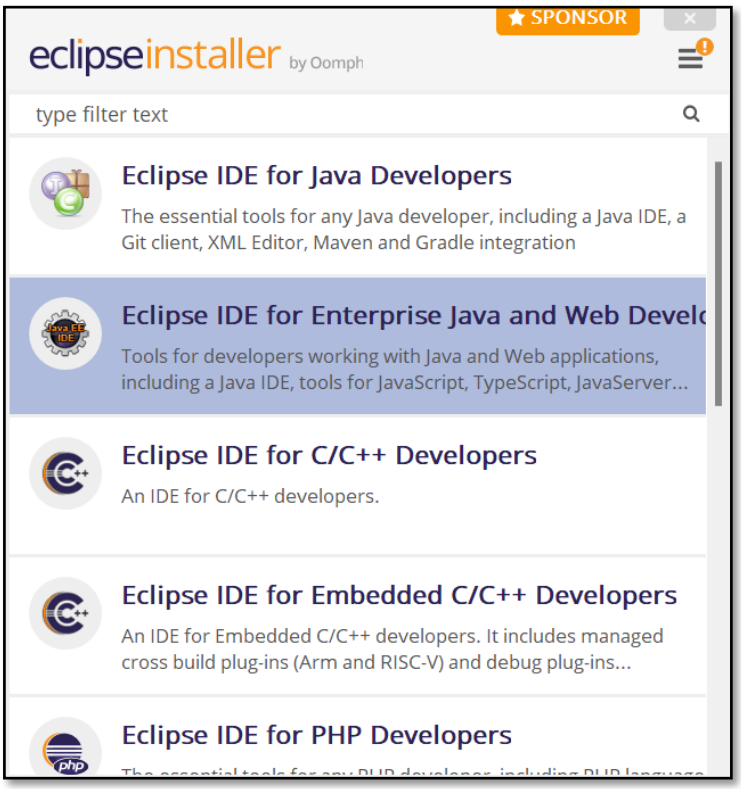
A. Importer un projet Maven à partir de « eclipse »

1. Télécharger, dans le dossier «**D:\Dev\Spring\Outils**», la dernière version de l'EDI «**eclipse**» à partir de l'URL suivante:

https://www.eclipse.org/downloads/



2. Double cliquer sur le fichier téléchargé pour installer eclipse avec la version JEE :

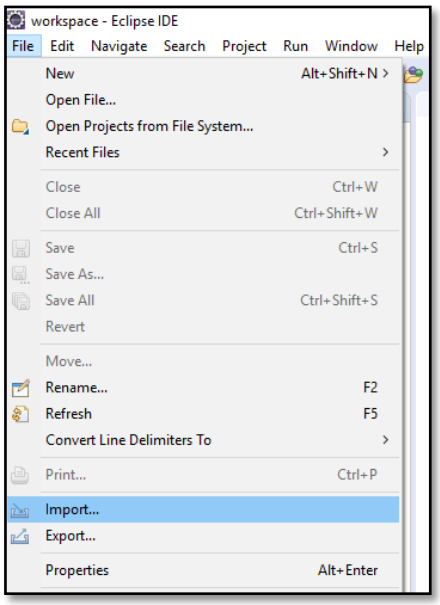


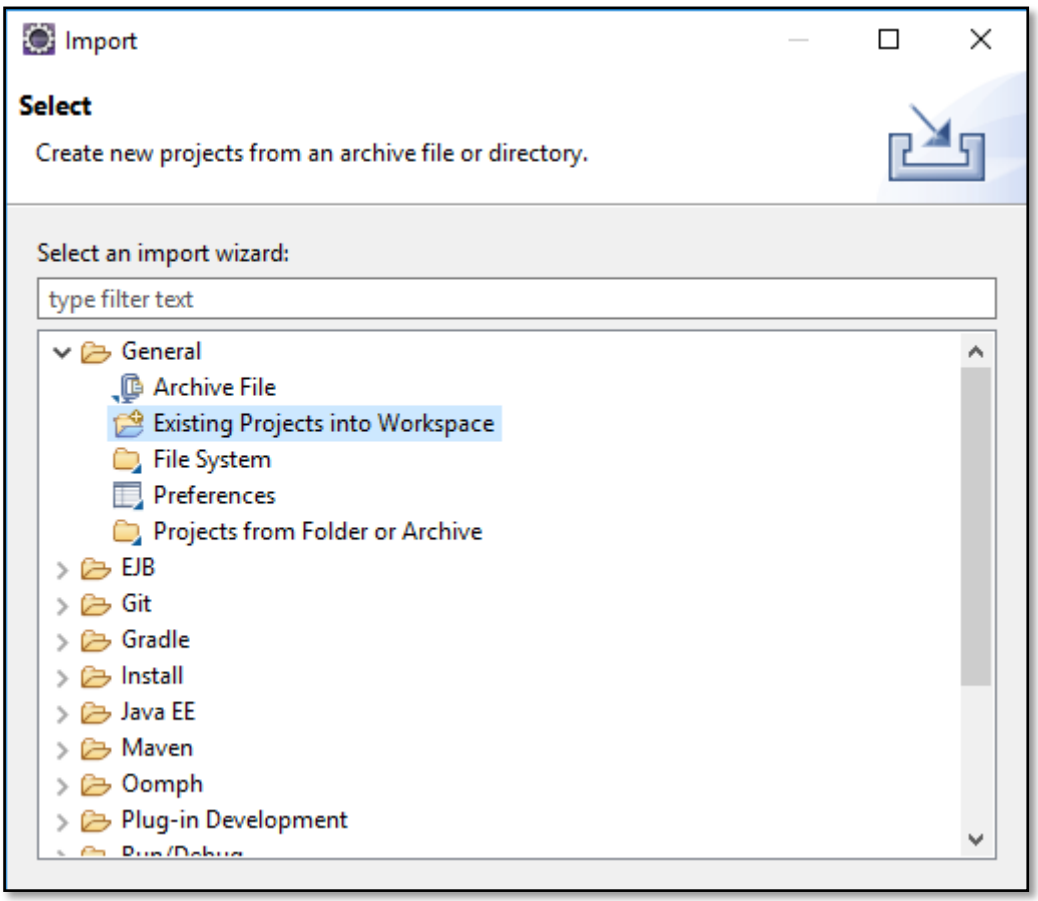
3. Choisir «**`D:\Dev\Spring\EDI\jee-2025-06`**» comme dossier d'installation de « eclipse » :

4. Ouvrir « **eclipse** ».
5. Indiquer le chemin du workspace (espace de travail):

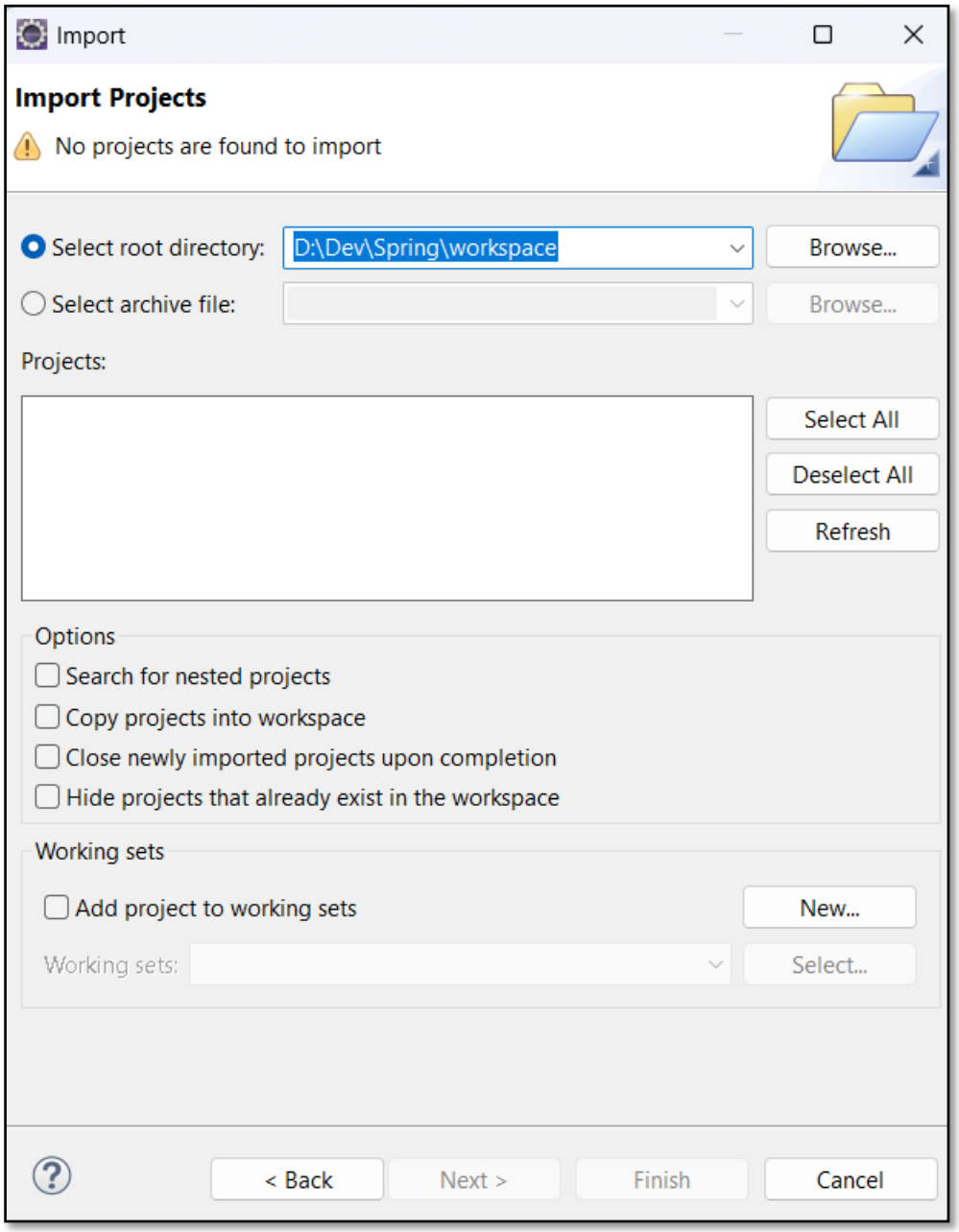
D:\Dev\Spring\workspace

6. Essayer d'importer le projet « **premierProjet**» dans eclipse (créé dans la partie 2 de l'atelier 01).



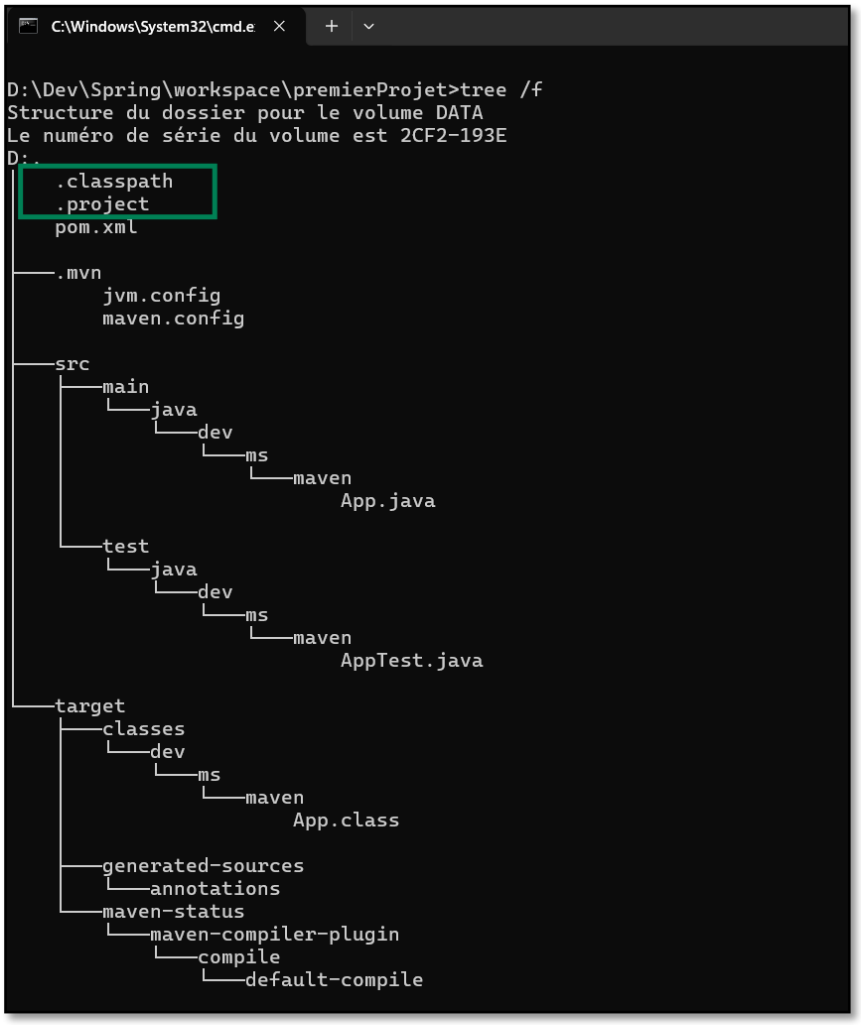


Cliquer sur « **Next** » et sélectionner le dossier « **workspace** » dans lequel existe le projet « **premierProjet** » :

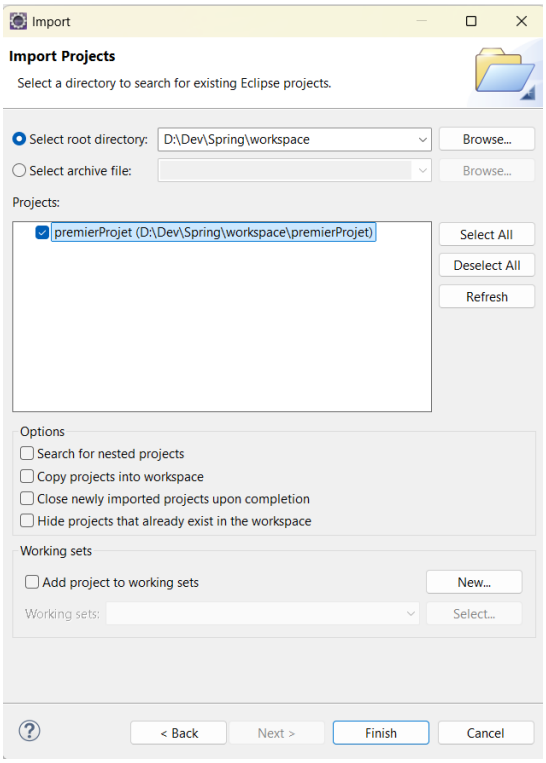


*Remarquer que « **eclipse** » n'arrive pas à importer ce projet qui n'a pas les caractéristiques d'un projet **eclipse (n'est pas construit avec l'outil « eclipse)** ; il ne contient pas les fichiers descripteurs « **.project** » et « **.classpath** ».*

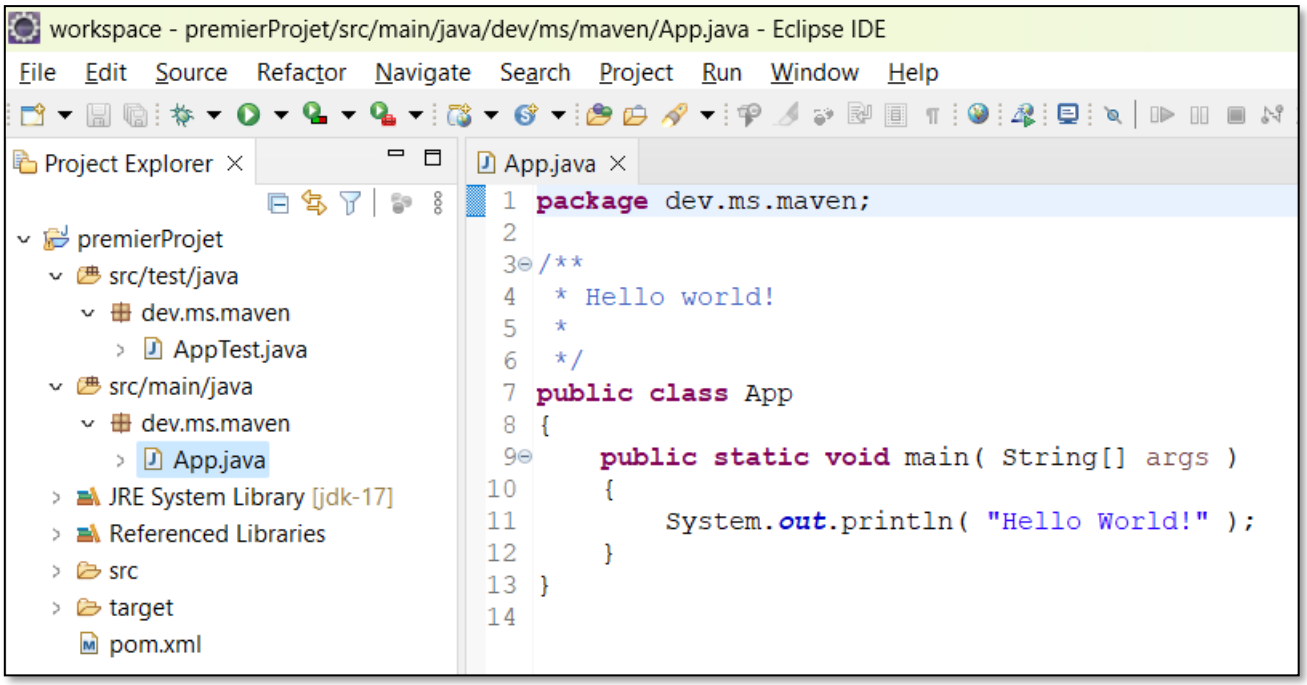
7. Pour se faire, on retourne à l'outil « **Maven** » en mode console :
   - ✓ Accèder au dossier « **premierProjet**».
   - ✓ Taper la commande **`mvn eclipse:eclipse`** pour rendre le projet de type « **eclipse** ».
   - ✓ Taper la commande **`tree /f`** et remarquer la génération des deux fichiers :
     - *.classpath*
     - *.project*



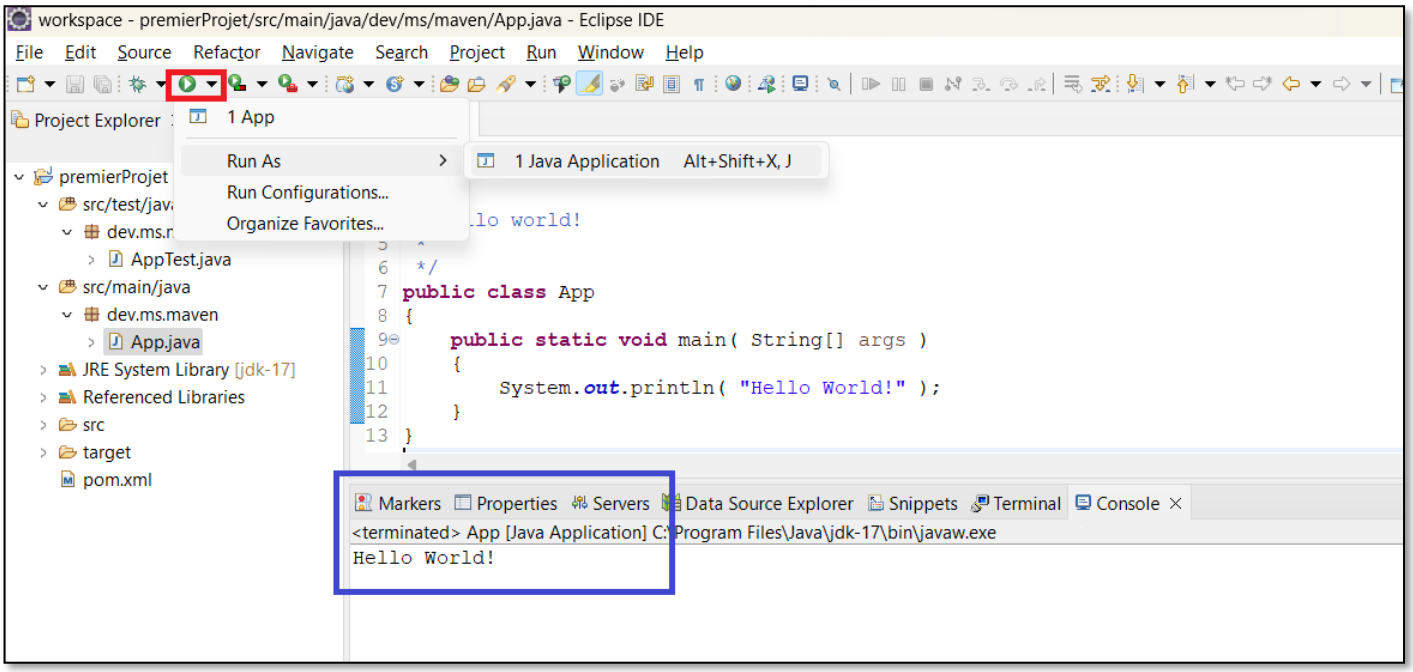
✓ Refaire maintenant la procédure d'importation dans **eclipse**. Remarquer, cette fois, la détection du projet « **premierProjet** » :



✓ Visualiser, maintenant, la structure du projet dans l'EDI « **eclipse** » :

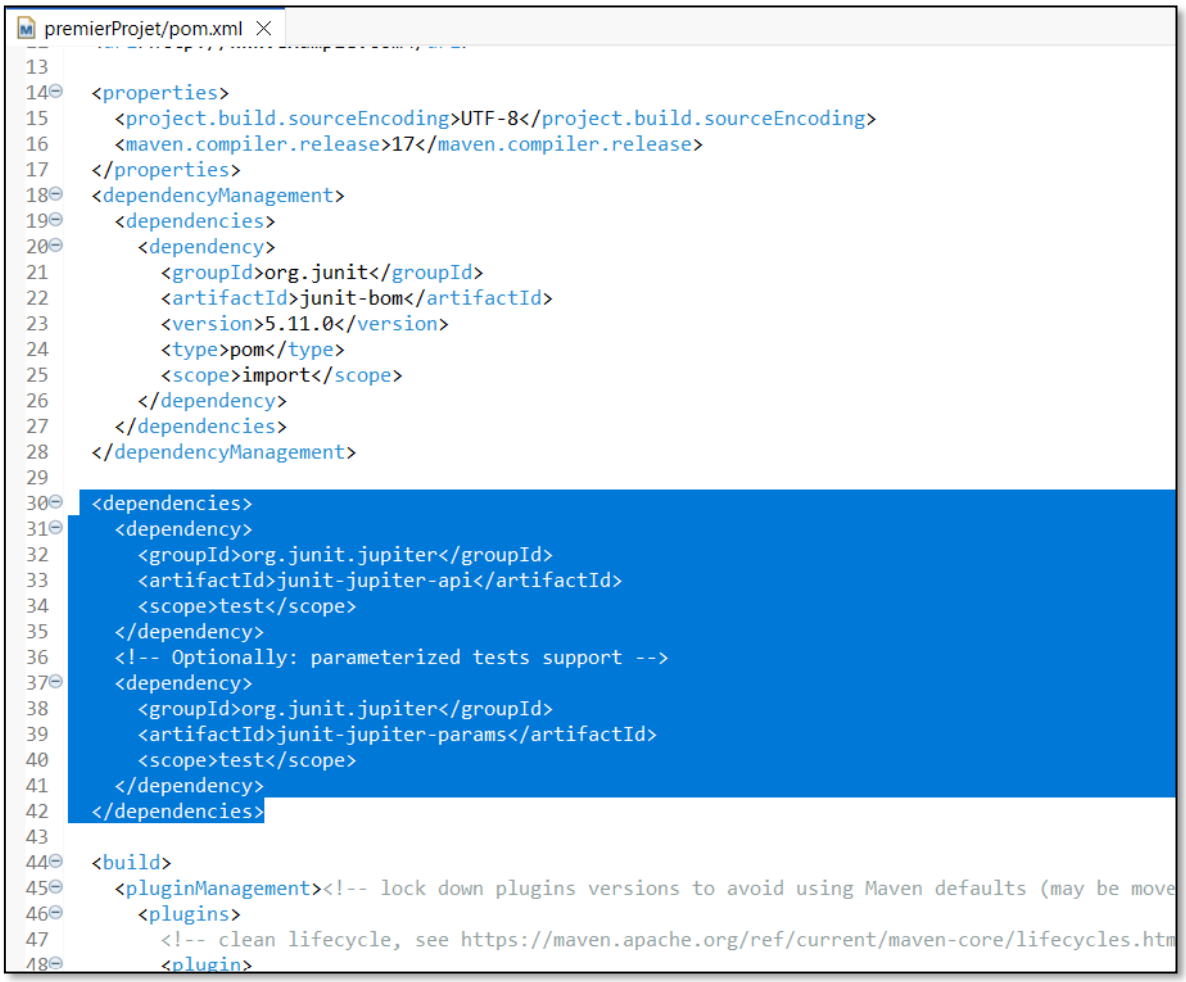


✓ Exécuter le programme avec eclipse :



B. Utiliser des commandes Maven (cycle de vie)

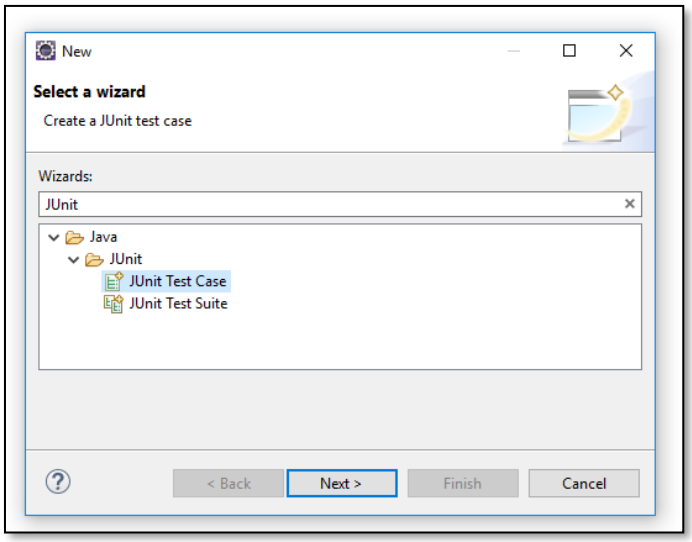
1. Consulter le fichier « **pom.xml** » et visualiser sa structure : La seule dépendance est **JUnit utilisée pour réaliser les tests unitaires** :



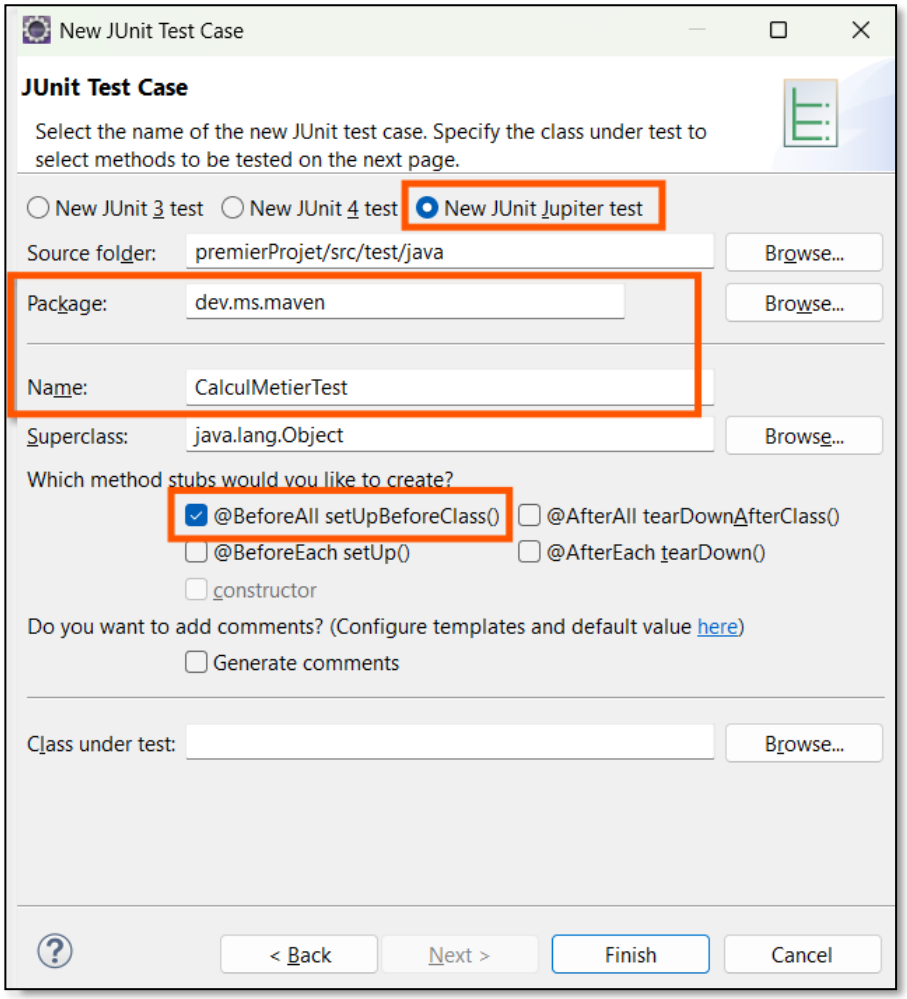
2. Supprimer les deux classes « **App.java** » et « **AppTest.java** » (inutiles pour la suite)
3. Supprimer aussi le sous-dossier « **target** »
4. Créer sous « **src/main/java** » et dans le package « **dev.ms.maven** » la classe « **CalculMetier** » ayant le code suivant :

```
package dev.ms.maven;

public class CalculMetier
{
    public double somme(double a, double b)
    {
        return a + b;
    }

    public double produit(double a, double b)
    {
        return a * b;
    }
}
```

5. Sélectionner le package « **dev.ms.maven** » sous « **src/test/java** » puis choisir la commande « **New/Other/JUnit Test Case** » pour créer une classe de test JUnit afin de tester le bon fonctionnements des méthodes de la classe «**CalculMetier**»:



6. Nommer la classe « **CalculMetierTest** » (prendre la version 4 de JUnit) et demander la génération de la méthode « **setUpBeforeClass()** » :

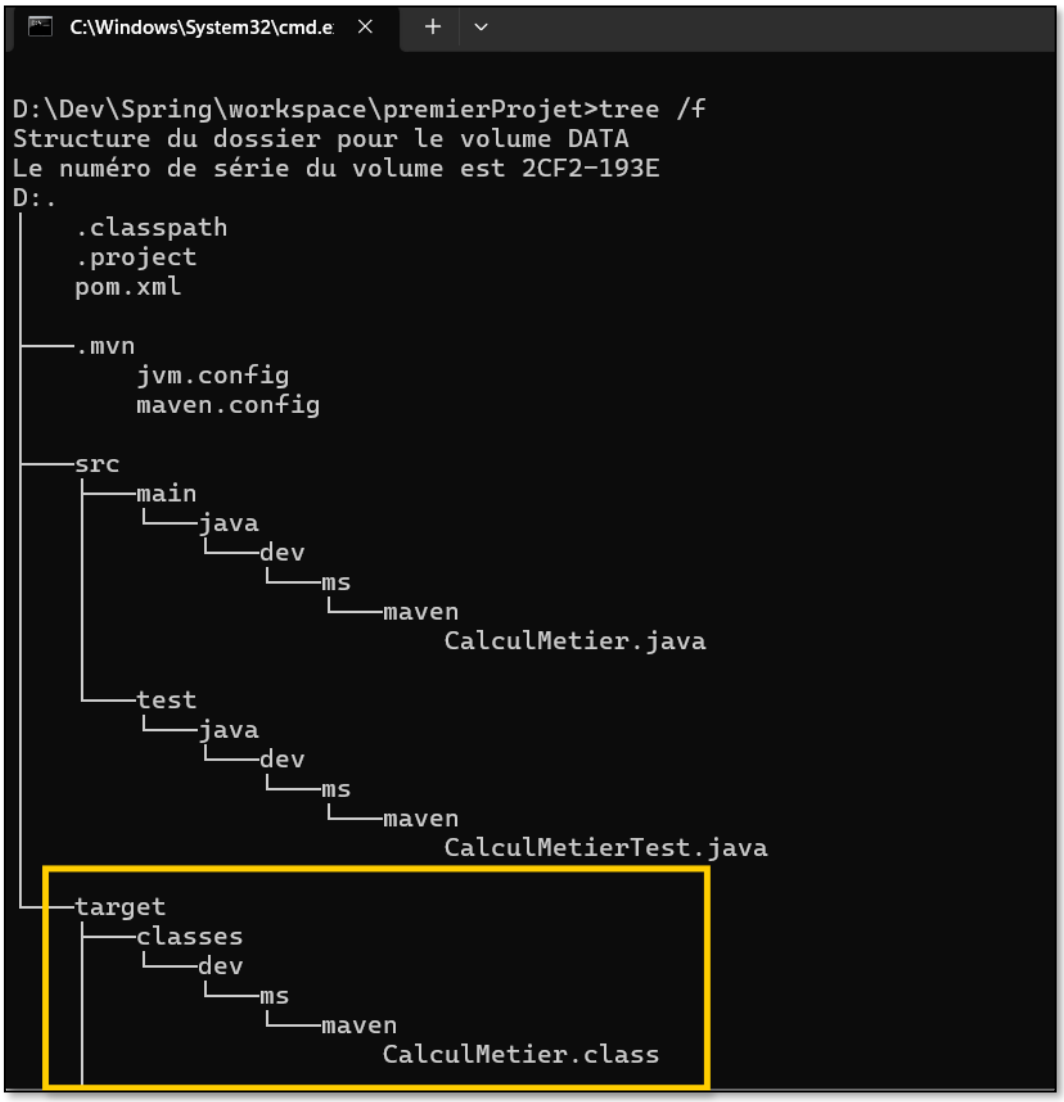


7. Cliquer sur « **Finish** » pour terminer la création de la classe de test puis remplacer le code généré par le code suivant :

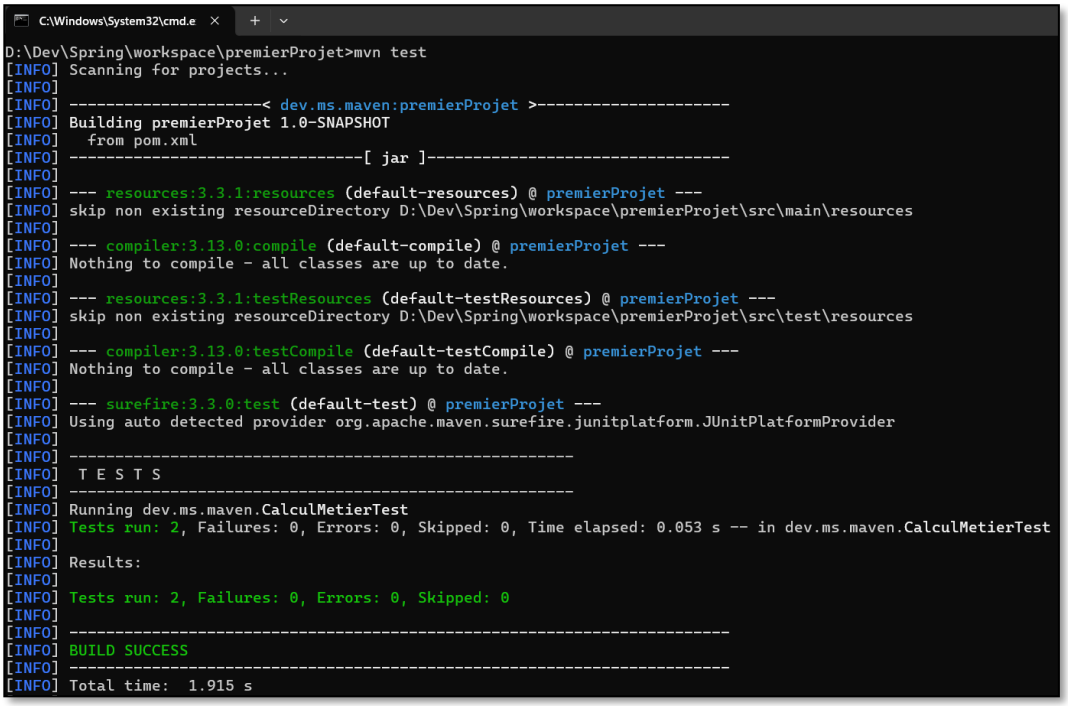
```java
package dev.ms.maven;

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

import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;

public class CalculMetierTest {
	//Déclarer une reference vers la classe CalculMetier
	private static CalculMetier metier;

	@BeforeAll
	public static void setUpBeforeClass() throws Exception {
    //Instancier la classe CalculMetier
		metier = new CalculMetier();
	}

	@Test
	public void testSomme()
	{
		assertTrue(metier.somme(5, 2) == 7);
	}

	@Test
	public void testProduit()
	{
		assertTrue(metier.produit(5, 2) == 10);
	}

}
```

- **La méthode « setUp()BeforeClass » est exécutée avant tous les tests de la classe : Elle est annotée avec @BeforeAll**
- **Toutes les méthodes de test sont annotées par @Test**
- **La méthode assertTrue(Boolean condition) est une méthode statique de la classe « org.junit.jupiter.api.Assertions» et indique que le test est correct si la condition est correcte.**

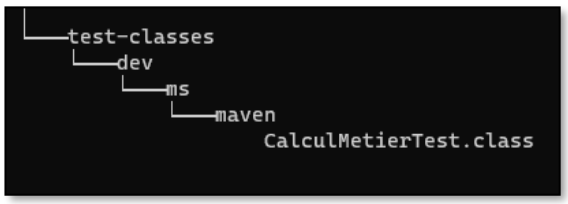
8. Passer à l'invite de commande et accéder au dossier « **premierProjet**» pour **compiler** le projet à travers la commande : **mvn compile** :
9. Taper la commande **tree /f** et remarquer la génération de la classe précompilée (fichier « **CalculMetier.class** ») dans le sous-dossier « **target\classes** » (compilation uniquement des fichiers sources et non des fichiers de test) :



10. **Tester** le projet à travers la commande **mvn test** (qui utilise la bibliothèque « **JUnit** » :



11. Le test a réussi : Remarquer la génération de la classe précompilée (le fichier «**CalculMetierTest.class**») dans le sous-dossier «**target\test-classes**» (utiliser toujours la commande **tree /f**)



12. Modifier la classe « **CalculMetier** » pour avoir un fonctionnement erroné des méthodes « **somme** » et « **produit** » puis relancer la compilation et le test. Prendre l'exemple suivant :

```java
package dev.ms.maven;
public class CalculMetier
{
    public double somme(double a, double b)
    {
        return a - b; // au lieu de a+b
    }

    public double produit(double a, double b)
    {
        return a / b;  // au lieu de a*b
    }
}
```

Pas de problème au niveau de compilation. Mais, au niveau de la phase de test, deux erreurs ont été déclenchées :

```
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running dev.ms.maven.CalculMetierTest
[ERROR] Tests run: 2, Failures: 2, Errors: 0, Skipped: 0, Time elapsed: 0.027 s <<< FAILURE! - in dev.ms.maven.CalculMetierTest
[ERROR] testSomme(dev.ms.maven.CalculMetierTest)  Time elapsed: 0.004 s  <<< FAILURE!
java.lang.AssertionError
        at dev.ms.maven.CalculMetierTest.testSomme(CalculMetierTest.java:21)

[ERROR] testProduit(dev.ms.maven.CalculMetierTest)  Time elapsed: 0 s  <<< FAILURE!
java.lang.AssertionError
        at dev.ms.maven.CalculMetierTest.testProduit(CalculMetierTest.java:27)

[INFO]
[INFO] Results:
[INFO]
[ERROR] Failures:
[ERROR]   CalculMetierTest.testProduit:27
[ERROR]   CalculMetierTest.testSomme:21
[INFO]
[ERROR] Tests run: 2, Failures: 2, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
```

13. Reprendre la version correcte de la classe « **CalculMetier** » et refaire la compilation et le test.

14. Ajouter une nouvelle classe «**MonApp**» dans le package «**dev.ms.maven**» qui permet d'appeler les deux méthodes de la classe «**CalculMetier**» :

```
package dev.ms.maven;
public class MonApp {
     public static void main(String[] args) {

          if (args.length<2)
          {
               System.out.println("Veuillez saisir deux nombres..");
               System.exit(0);
          }
          double a = Double.parseDouble(args[0]);
          double b = Double.parseDouble(args[1]);
          CalculMetier cm =new CalculMetier();
          double s =cm.somme(a, b);
          double p = cm.produit(a, b);
          System.out.println("La somme de "+a+ "et "+b+ "est: "+s);
          System.out.println("Le produit de "+a+ "et "+b+ "est: "+p);
     }
}
```

15. Compiler le projet et lancer l'exécution à travers la commande suivante :

```
java -cp .\target\classes dev.ms.maven.MonApp
```

a) En passant 0 arguments :

```
C:\Windows\System32\cmd.e
D:\Dev\Spring\workspace\premierProjet>java -cp .\target\classes dev.ms.maven.MonApp
Veuillez saisir deux nombres..
```

b) En passant 1 seul argument :

```
C:\Windows\System32\cmd.e
D:\Dev\Spring\workspace\premierProjet>java -cp .\target\classes dev.ms.maven.MonApp 4
Veuillez saisir deux nombres..
```

c) En passant 2 arguments :

```
C:\Windows\System32\cmd.e
D:\Dev\Spring\workspace\premierProjet>java -cp .\target\classes dev.ms.maven.MonApp 4 5
La somme de 4.0et 5.0est: 9.0
Le produit de 4.0et 5.0est: 20.0
```

NB : Il est possible d'exécuter le projet autrement avec une commande mvn en utilisant un plugin «exec-maven-plugin» (les plugins seront traités en détail dans la partie05 de cet atelier) Pour se faire, ajouter à l'intérieur de la balise « Build » du fichier « pom.xml » la déclaration du plugin comme suit :

```
<plugins>

        <plugin>
            <groupId>org.codehaus.mojo</groupId>
            <artifactId>exec-maven-plugin</artifactId>
            <version>3.1.0</version>
            <configuration>
                <mainClass>dev.ms.maven.MonApp</mainClass>
            </configuration>
        </plugin>
</plugins>
```

Enregistrer puis lancer la commande suivante :

mvn exec:java

Le programme est exécuté mais nécessite le passage des arguments :

```
D:\Dev\Spring\workspace\premierProjet>mvn exec:java
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< dev.ms.maven:premierProjet >----------------------
[INFO] Building premierProjet 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ premierProjet ---
Veuillez saisir deux nombres..
```

Modifier l'écriture de la commande pour passer deux nombres comme arguments d'exécution :

mvn exec:java -Dexec.args="4 5"

```
D:\Dev\Spring\workspace\premierProjet>mvn    exec:java   -Dexec.args="4  5"
[INFO] Scanning for projects...
[INFO]
[INFO] ----------------------< dev.ms.maven:premierProjet >----------------------
[INFO] Building premierProjet 1.0-SNAPSHOT
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- exec:3.1.0:java (default-cli) @ premierProjet ---
La somme de 4.0et 5.0est: 9.0
Le produit de 4.0et 5.0est: 20.0
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.579 s
```

16. Passer, maintenant, à archiver le projet : càd **packager** le projet à travers la commande :

mvn package

Remarquer la création d'un package d'archive «**premierProjet-1.0-SNAPSHOT .jar** »

dans le sous-dossier « **target** » :

```
[INFO] Building jar: D:\Dev\Spring\workspace\premierProjet\target\premierProjet-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  23.033 s
```

17. Essayer le lancer l'exécution du programme à travers le fichier d'archive nouvellement créé à travers la commande suivante :

```
java -jar .\target\premierProjet-1.0-SNAPSHOT.jar
```

Une erreur est déclenchée pour signaler qu'il est impossible de détecter une classe de démarrage ayant une méthode main :

```
C:\Windows\System32\cmd.e
D:\Dev\Spring\workspace\premierProjet>java -jar .\target\premierProjet-1.0-SNAPSHOT.jar
no main manifest attribute, in .\target\premierProjet-1.0-SNAPSHOT.jar
```

18. Pour remédier à ce problème, éditer le fichier « **pom.xml** » et ajouter une configuration pour le plugin « **jar** » pour spécifier la classe main (classe de démarrage) comme suit avec le caractère en gras :

```
<plugin>
          <artifactId>maven-jar-plugin</artifactId>
          <version>3.4.2</version>

          <configuration>
                    <archive>
                              <manifest>
                                        <mainClass>dev.ms.maven.MonApp</mainClass>
                              </manifest>
                    </archive>
          </configuration>

</plugin>
```

19. **Repackager** le projet puis **resaisir** la commande :

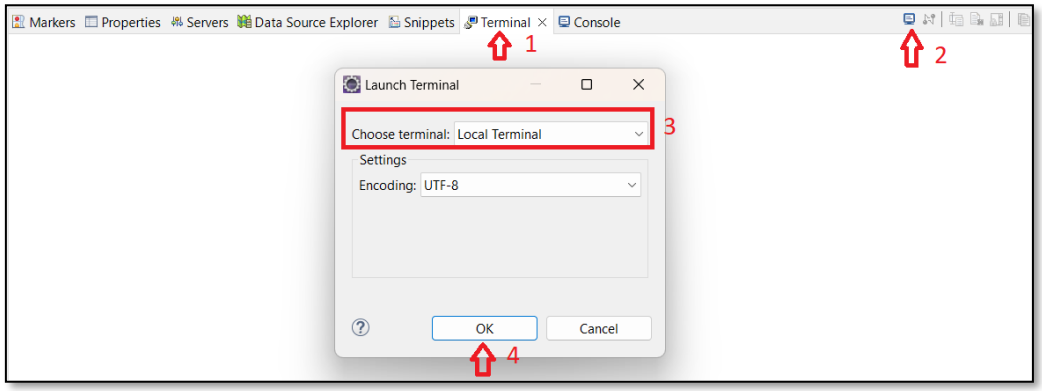
```
java -jar .\target\premierProjet-1.0-SNAPSHOT.jar
```

et remarquer l'exécution du programme

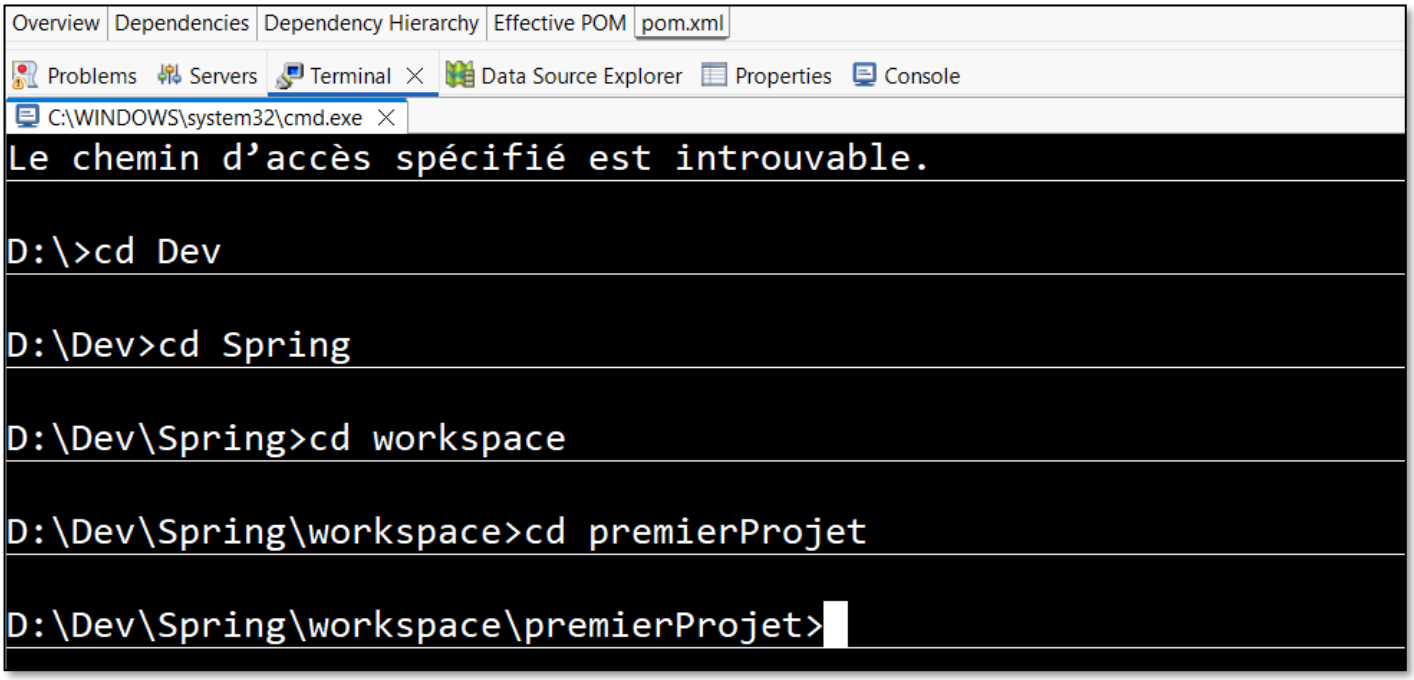
```
C:\Windows\System32\cmd.e

D:\Dev\Spring\workspace\premierProjet>java   -jar   .\target\premierProjet-1.0-SNAPSHOT.jar 7 5
La somme de 7.0et 5.0est: 12.0
Le produit de 7.0et 5.0est: 35.0
```

20. Passons à **publier** (ou bien installer) le projet dans le dépôt local de **Maven.**

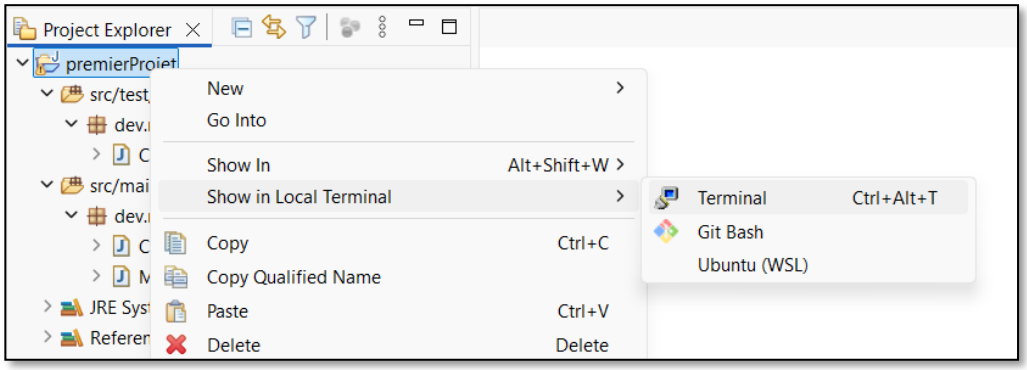
a) Cette fois utiliser le Terminal de «**eclipse** » :



b) Se positionner dans le dossier de notre projet lorsque le terminal s'affiche :



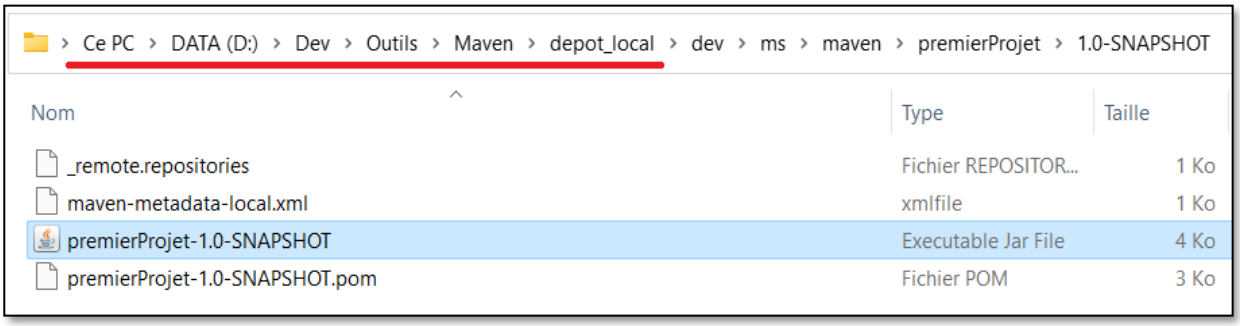
Ou bien, tout court, sélectionner le nom du projet et choisir la commande : « Show in Local terminal » :



c) Utiliser la commande `mvn install` et remarquer le stockage du paquetage du projet « **premierProjet** » dans le repository local :

```
[INFO] Installing D:\Dev\Spring\workspace\premierProjet\target\premierProjet-1.0-SNAPSHOT.jar to D:
Dev\Outils\Maven\depot_local\dev\ms\maven\premierProjet\1.0-SNAPSHOT\premierProjet-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  8.649 s
```

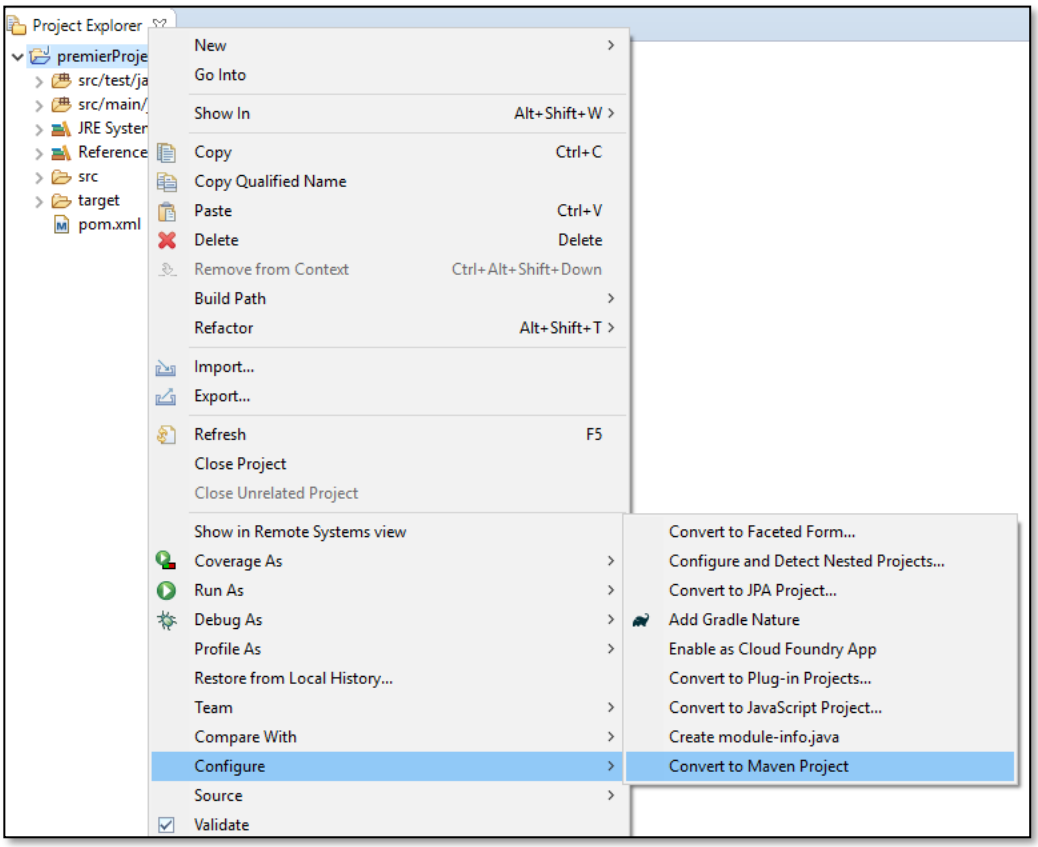
d) Le projet est ainsi livré dans le dépôt local :



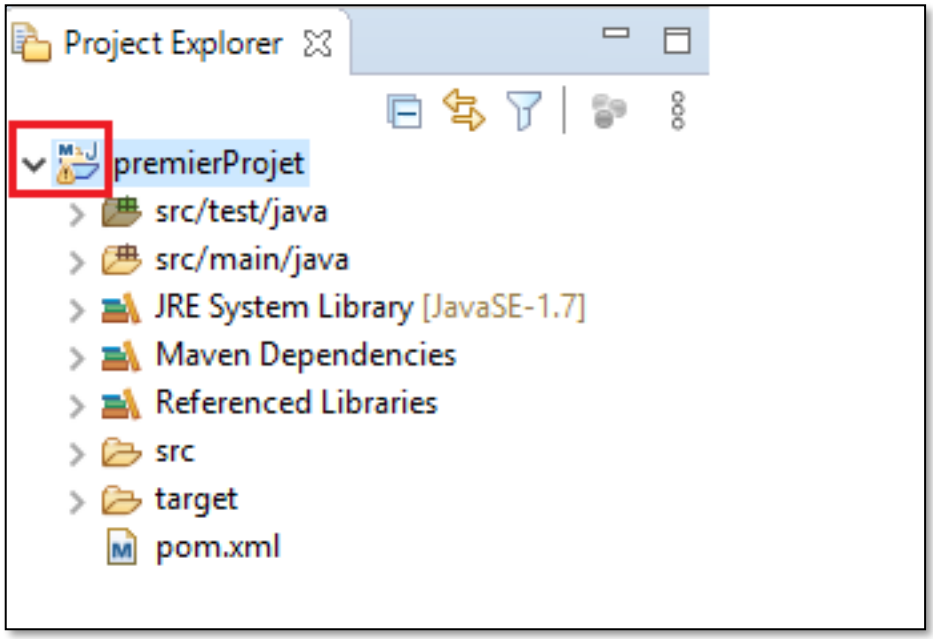
C. Utiliser les modules de « eclipse » pour passer des commandes Maven avec l'assistant graphique

21. Convertir le projet « **premierProjet** » en un projet « **Maven** ». Ainsi, on intègre l'outil « **Maven** » dans l'EDI « **eclipse** » et on manipule les commandes « **Maven** » graphiquement à partir des modules « **eclipse** ». Pour se faire, suivre les étapes suivantes :

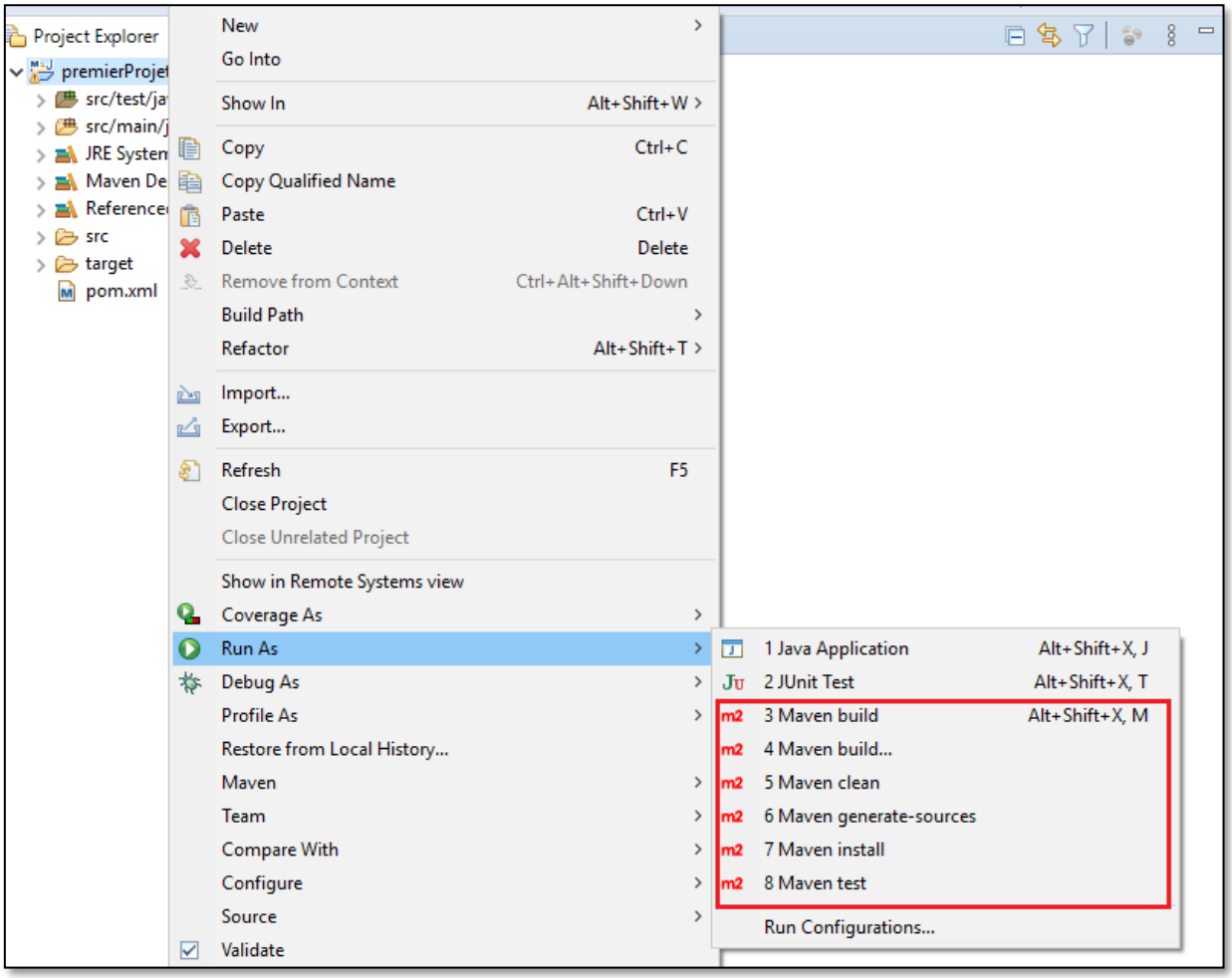
a) Sélectionner le projet et choisir la commande « **Convert to Maven Project** » :



b) Remarquer l'étiquette « **M** » ajoutée sur le nom du projet indiquant qu'il s'agit désormais d'un projet **Maven** :



22. Pour utiliser une commande « **Maven** », il suffit de sélectionner le projet et choisir la commande « **Run As** » pour afficher une liste de commandes :



23. Lancer, graphiquement, les commandes « **Maven** » pour retrouver les mêmes résultats.