ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Introduction et consignes
TP Java n°4

A. Lebret, 2025

Tester, tester, tester

Tester avec JUnit 5

- JUnit 5 est la bibliothèque de test standard pour Java
- Elle permet d’écrire des tests unitaires simples, lisibles et maintenables
- JUnit 5 se compose de plusieurs modules (JUnit Jupiter, JUnit Platform, etc.).

Page officielle : **https://junit.org/junit5/**

Annotations de base

- **`@Test`** : Indique qu'une méthode est un cas de test
- **`@BeforeEach`** : préambule exécuté avant chaque test, utile pour initialiser l'état
- **`@AfterEach`** : postambule exécuté après chaque test, pour nettoyer les ressources
- **`@BeforeAll`** et **`@AfterAll`** : exécutés une fois avant/après tous les tests (méthodes statiques)

Exemple

```
@BeforeEach
void setUp() {
    // Initialisation avant chaque test
}

@Test
void testSomething() {
    // Un cas de test
}
```

Assertions et vérifications

Les assertions permettent de vérifier les résultats attendus :

- **`assertEquals(expected, actual)`** : vérifie l'égalité
- **`assertTrue(condition)`** et **`assertFalse(condition)`** : vérifient une condition
- **`assertNotNull(object)`** : vérifie qu'un objet n'est pas nul
- **`assertThrows()`** : vérifie qu'une exception est levée

Exemple

```
@Test
void testSum() {
    int sum = calculateSum(2, 3);
    assertEquals(5, sum, "2 + 3 should be equal to 5");
}
```

Utilisation de @TempDir

- **`@TempDir`** permet de créer un répertoire temporaire pour les tests
- Idéal pour tester des opérations sur fichiers sans affecter le système

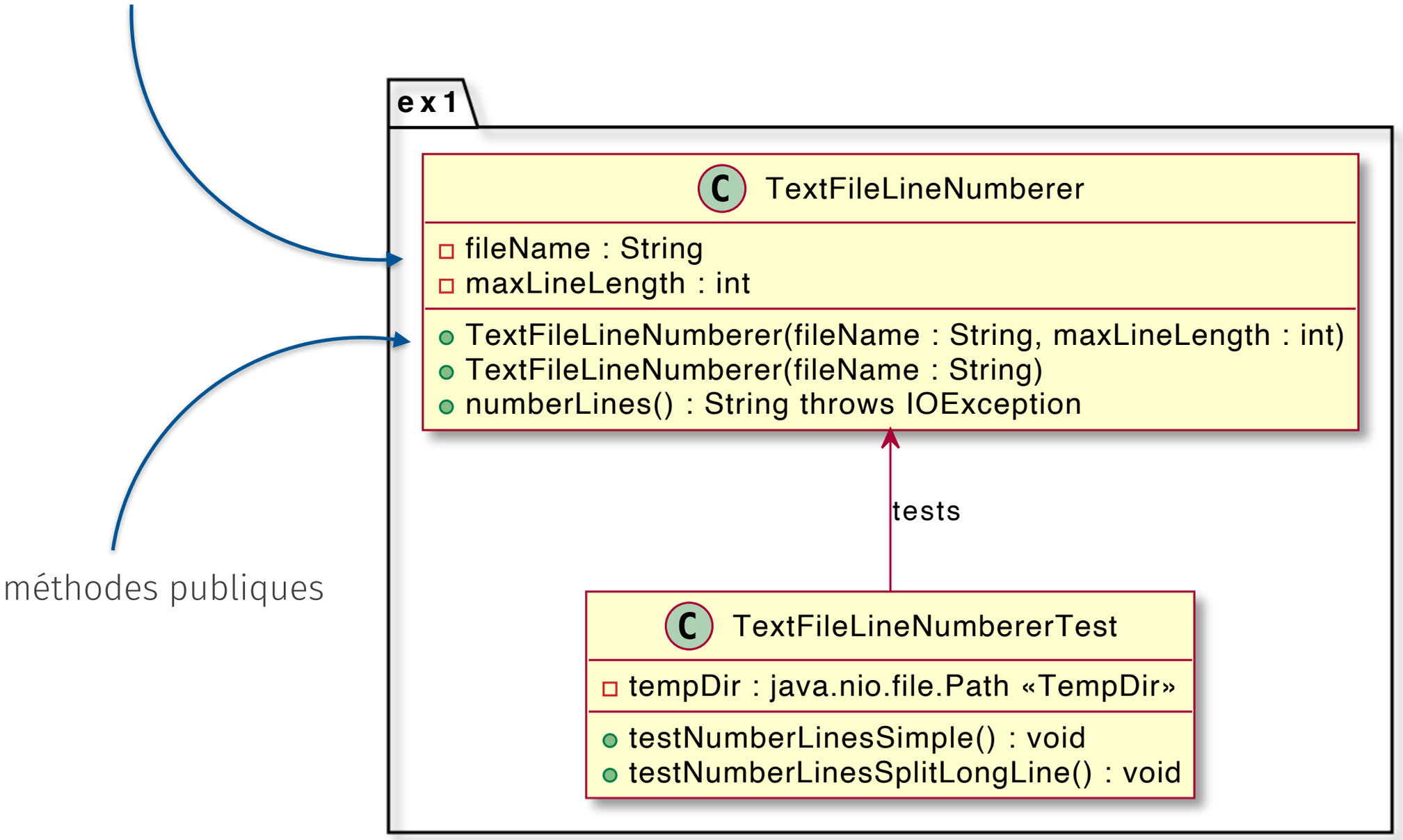
Exemple

```
@TempDir
Path tempDir;

@Test
void testFileCreation() throws IOException {
    Path tempFile = tempDir.resolve("test.txt");
    Files.write(tempFile, "Bonjour".getBytes());
    assertTrue(Files.exists(tempFile));
}
```

Exercice n°1

Diagramme de classes

attributs privés



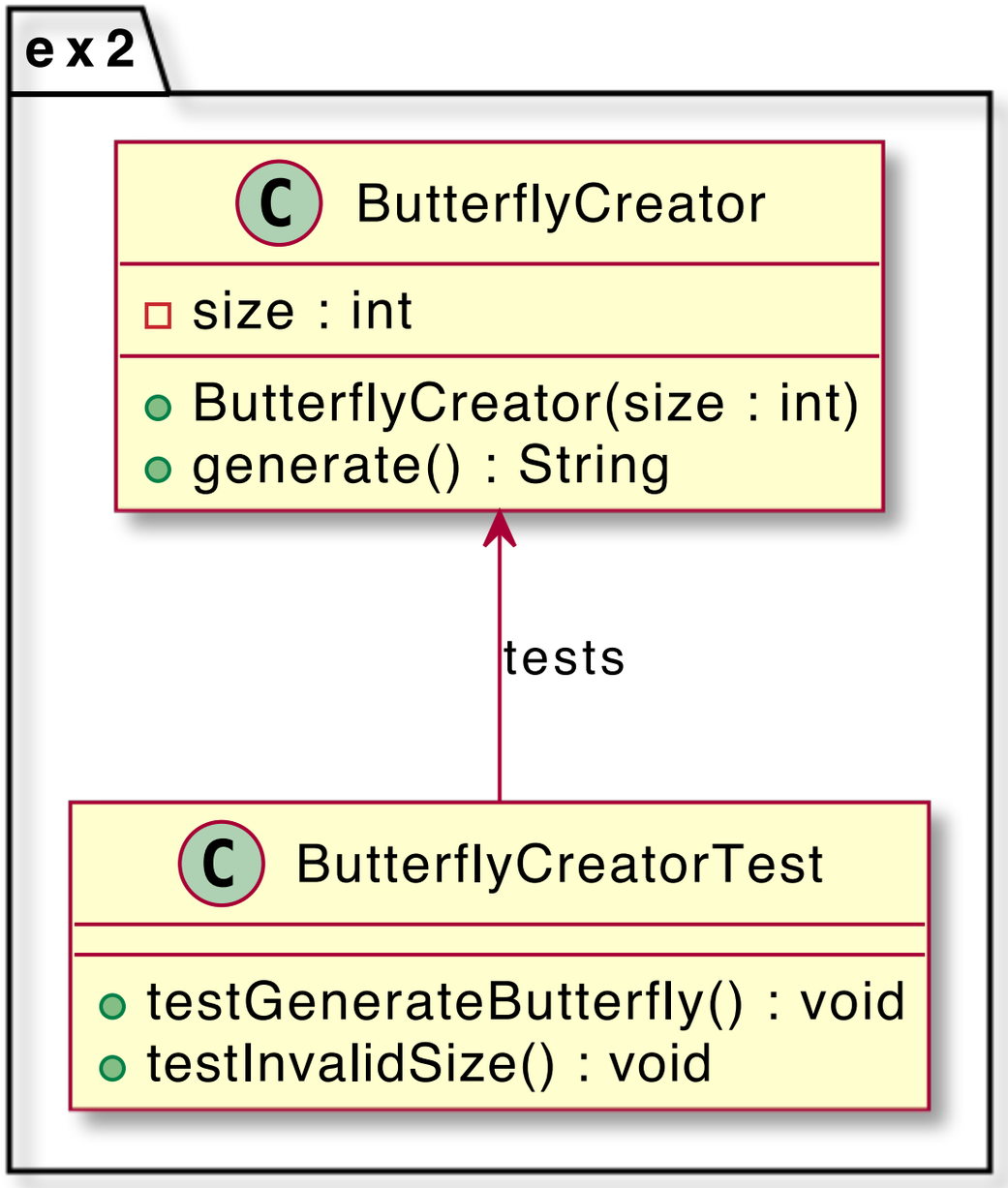
méthodes publiques

Test de la classe TextFileLineNumberer

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.io.TempDir;

@TempDir
Path tempDir;

@Test
public void testNumberLinesSimple() throws IOException {
    // Création d'un fichier temporaire
    Path tempFile = tempDir.resolve("test.txt");
    String content = "Ligne 1\n\nLigne 3";
    Files.write(tempFile, content.getBytes(StandardCharsets.UTF_8));

    TextFileLineNumberer numberer = new
                                TextFileLineNumberer(tempFile.toString(), 60);
    String result = numberer.numberLines();

    String expected = String.format("%4d: %s\n", 1, "Ligne 1") +
                      String.format("%4d: %s\n", 2, "") +
                      String.format("%4d: %s\n", 3, "Ligne 3");
    assertEquals(expected, result);
}
```

Exercice n°2

Diagramme de classes

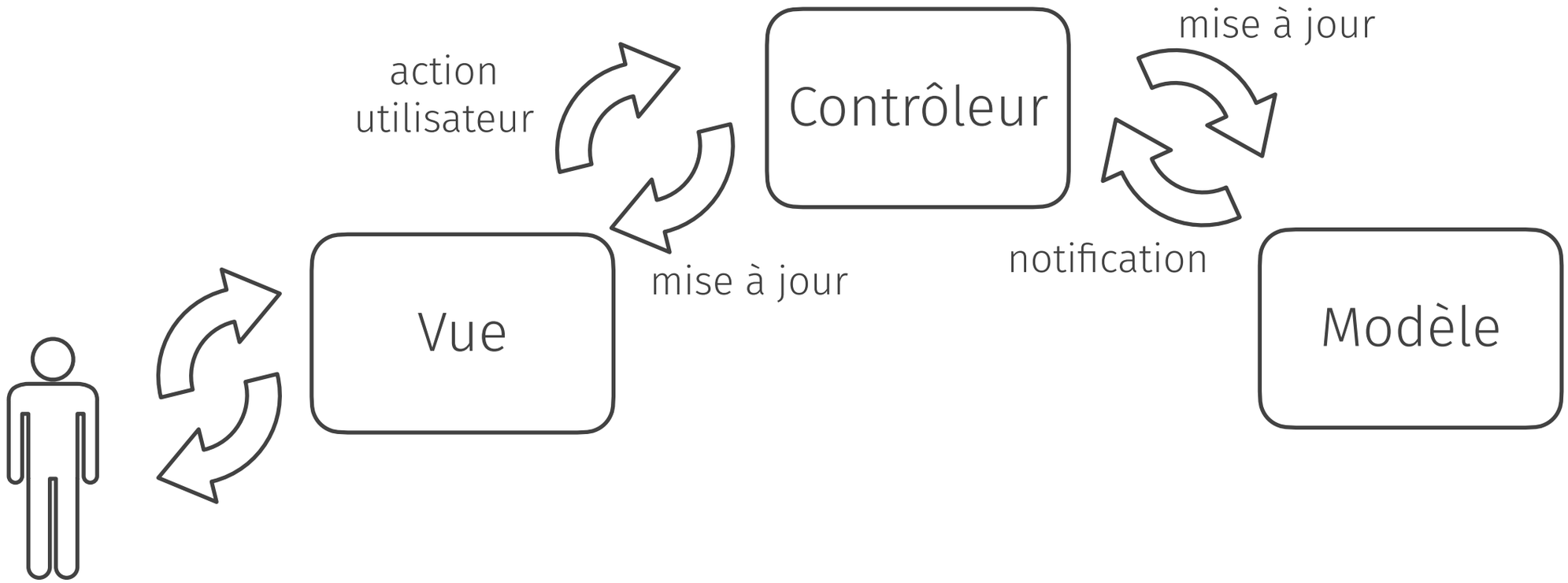


Exercice n°3

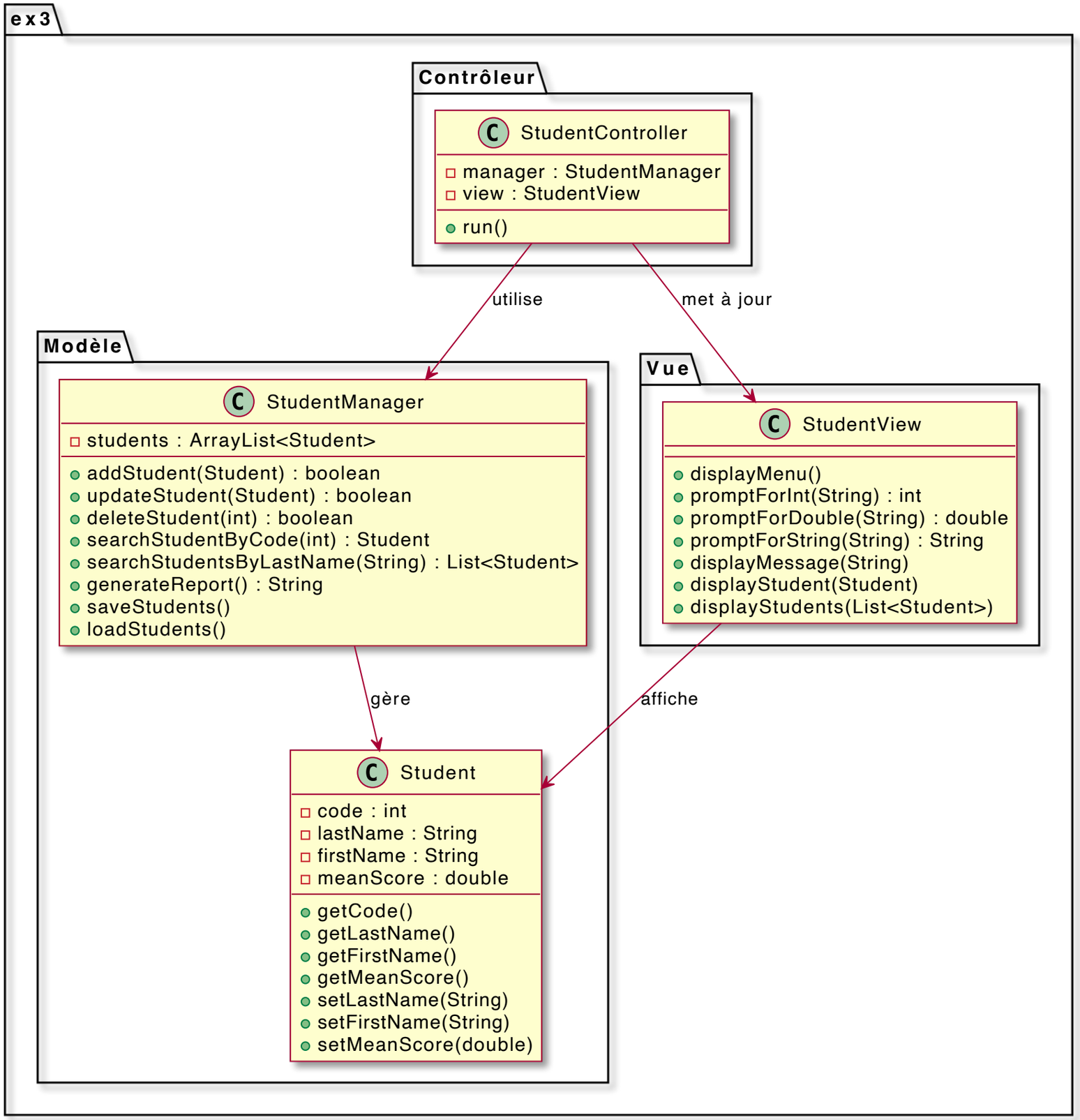
À propos du patron MVC

MVC (modèle-vue-contrôleur) est un patron de conception qui sépare une application en trois composants interconnectés

- **Modèle** : gère les données et la logique métier
- **Vue** : s'occupe de l'affichage et de l'interface utilisateur
- **Contrôleur** : traite les entrées utilisateur et fait le lien entre modèle et vue



Application à l'exercice



À propos du patron MVC : le « modèle »

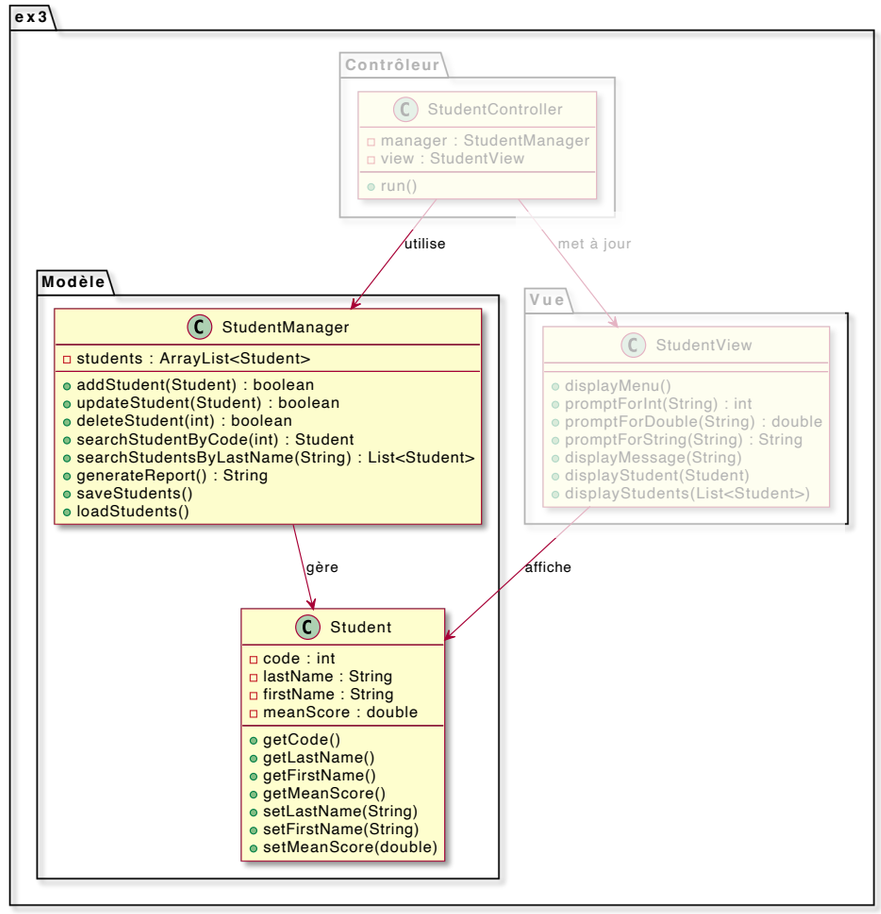
`Student`

- représente un étudiant avec des attributs tels que son code, son nom, son prénom et sa moyenne

`StudentManager`

- gère la liste des étudiants (ajout, modification, suppression, recherche, génération de rapport)

- Le **modèle** est indépendant de toute logique d'affichage et « isole » la logique métier

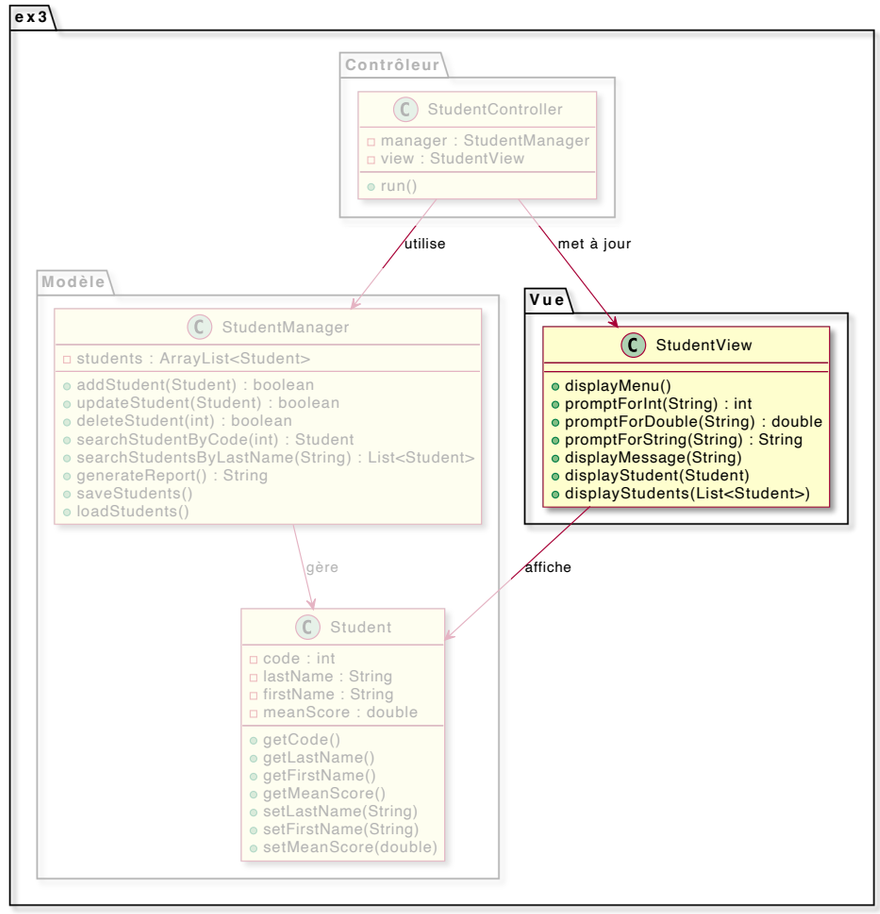


À propos du patron MVC : la « vue »

`StudentView`

- responsable des entrées/sorties console
- affiche les menus, demande des informations à l'utilisateur et affiche les attributs sur un étudiant

- La **vue** ne s'occupe que de la présentation et de la collecte des informations, sans connaître la logique métier

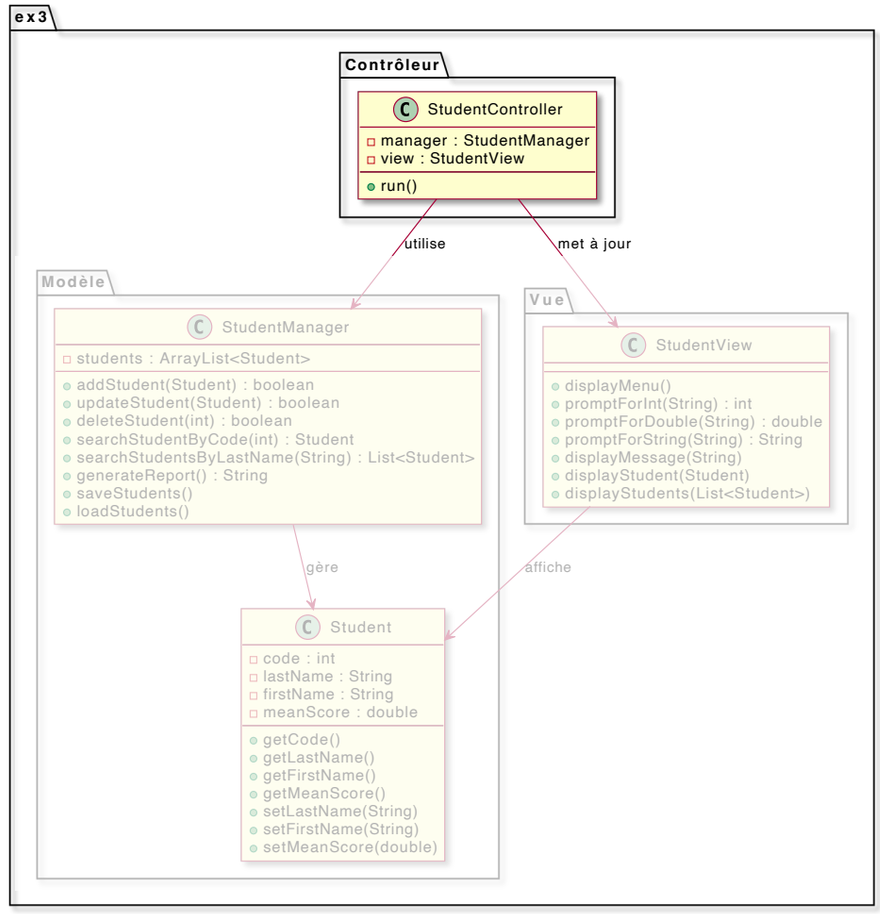


À propos du patron MVC : le « contrôleur »

`StudentController`

- sert d’intermédiaire entre le modèle et la vue
- charge les données, affiche le menu, traite les commandes de l'utilisateur et appelle les méthodes du modèle

- Le **contrôleur** orchestre l’application sans intégrer directement la persistance ou le formatage d’affichage



Exercice n° 4

Diagramme de classes