

Exercice 1 – Questions de cours (6 pts)

1. Qu'est-ce qu'un test paramétré en JUnit 5 ? (1 pt)
2. Citez deux annotations utilisées pour les tests paramétrés en JUnit 5 et expliquez leur rôle. (1 pt)
3. Quel est le rôle de `@BeforeEach` et `@AfterEach` dans un test unitaire ? (1 pt)
4. Dans un diagramme de cas d'utilisation, expliquez la différence entre `<<include>>` et `<<extend>>`. (1 pt)
5. Quelle est la différence entre un message synchrone et asynchrone dans un diagramme de séquence ? (1 pt)
6. Quelle est la différence principale entre un diagramme de cas d'utilisation et un diagramme de séquence ? (1 pt)

Exercice 2 – Diagramme de cas d'utilisation (7 pts)

Vous êtes chargé de modéliser un système de gestion des enseignants, des matières et des affectations pour un établissement scolaire.

Le système doit permettre :

✓ **L'administrateur de**

- Gérer les enseignants :
 - Ajouter, modifier, supprimer et rechercher un enseignant
 - Définir un enseignant comme *enseignant principal*
 - Consulter les affectations d'un enseignant
- Gérer les matières :
 - Ajouter, modifier, supprimer et consulter une matière
- Gérer les affectations : pour cela il doit
 - Affecter des matières aux enseignants après avoir vérifié leur disponibilité
 - Affecter une salle à chaque cours
- Générer des rapports sur les affectations

✓ **L'enseignant :**

- Consulter les matières qui lui sont attribuées

✓ **L'enseignant principal :** peut en plus

- Consulter les affectations détaillées de tous les enseignants
- Consulter des statistiques avancées

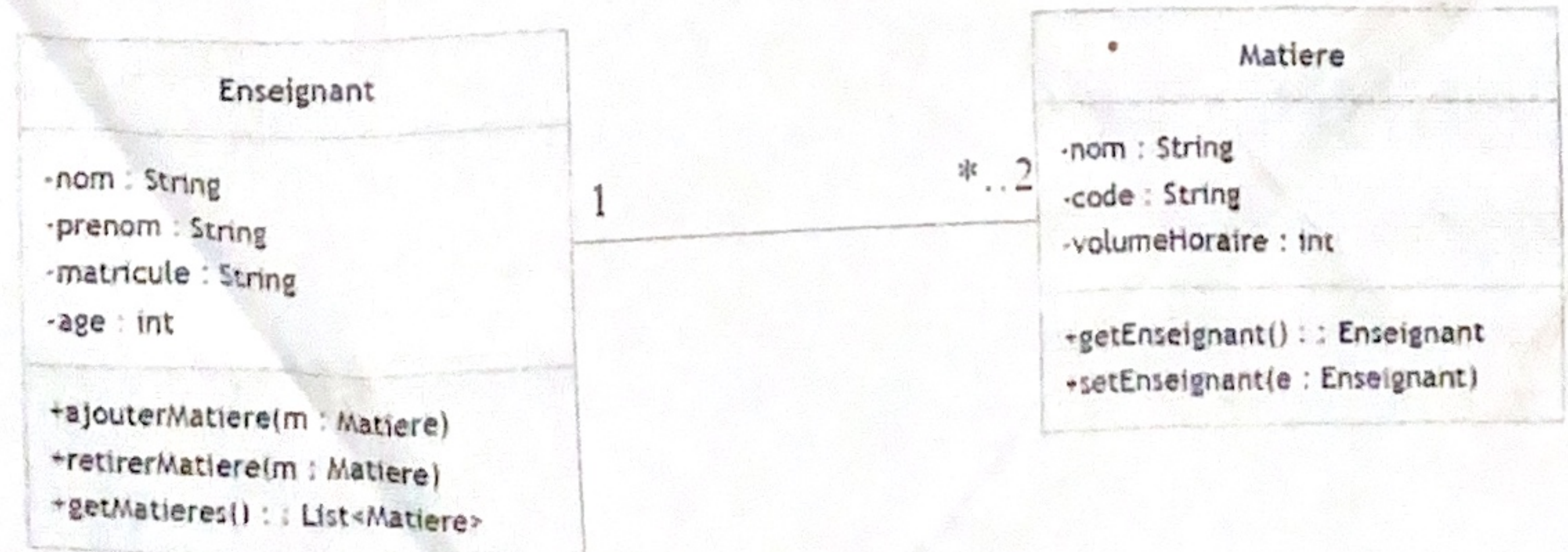
Travail demandé

1. Identifiez les acteurs du système
2. Identifiez les cas d'utilisation de chaque acteur
3. Dessinez un diagramme de cas d'utilisation UML

Exercice 3 : Test Junit

On se donne le diagramme de classes suivant :

(7 pts)



Instructions aux étudiants

Dans la classe de test Junit EnseignantTest ci-dessous pour tester la classe Enseignant :

- Vous n'avez pas à coder le diagramme de classes.
- Complétez **chaque test** pour vérifier le comportement attendu selon le diagramme de classes.
- Pour les exceptions, utilisez `assertThrows`.
- Pour les tests simples, utilisez `assertEquals` ou `assertTrue`.

On se donne une classe de test Junit incomplète :

```
class EnseignantTest {  
    Enseignant e;  
    Matiere m1, m2, m3;  
  
    @BeforeEach  
    void setup() {  
        e = new Enseignant("Moussa", "Bah", "111", 24);  
        m1 = new Matiere("GL", "DSI311", 24);  
    }  
}
```

1. Complétez le code de `testAjoutMatiere()` pour :

- Affecter deux matières à Moussa.
- Vérifiez que Moussa est affecté deux matières seulement
- Ajouter une 3eme matière et vérifier que l'exception est levée

```
@Test  
void testAjoutMatiere() {  
}
```

2. Complétez le code de `testConstructeurInvalide()` pour:

Lever une `IllegalArgumentException` dans chacun des cas suivants :

- Nom vide
- Prénom vide
- Age < 15 ou > 100

```
@Test  
void testConstructeurInvalide() {  
  
}
```

3. Complétez le code de `testRetirerMatiere()` pour :

Vérifiez que la liste de matières est correctement mise à jour après avoir retiré une matière à Moussa.

```
@Test  
void testRetirerMatiere() {  
  
}
```

Bonne chance !