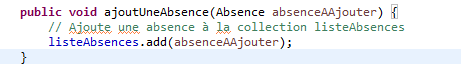
**Votre mission**

1. **Implémentez la méthode *ajoutUneAbsence* de la classe** Salarie

****

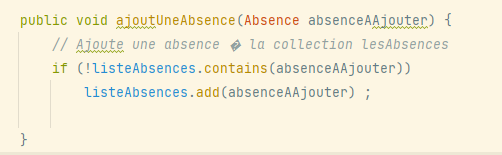
**Erreurs à éviter :**

**Affectation à un attribut absenceAAjouter qui n’existe pas**

**Affectation à une liste qui n’existe pas**

**On ne vous demande pas d’ajouter de nouvel attribut**

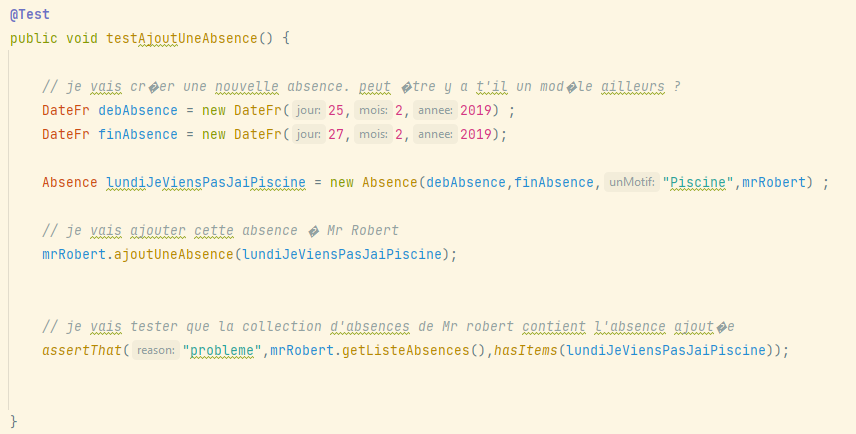
**Le top :**

****

**Eviter d’avoir 2 fois la même instance dans la collection si elle est déjà présente.**

**Cf cours + TP sur le sujet.**

1. **Ecrire un test unitaire dans la classe** TestSalarie **qui vérifie que la méthode *ajoutUneAbsence* fonctionne**

****

**Erreurs à éviter :**

**Perdre du temps ou se tromper en essayant de faire sans le modèle (TestDemandeAbsence)**

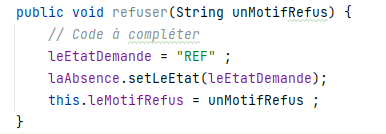
**Faire la préparation du test sans le test lui-même … (assertThat)**

**1ère partie : on crée une instance d’Absence, nommée lundiJeViensPasJaiPiscine. On a besoin de 2 dates. On a trouvé et repris un modèle dans la classe de test existante TestDemandeAbsence**

**2ème partie : j’applique la méthode codée (ajouteUneAbsence) à mon instance de Salarie disponible (mrRobert, instancié dans le BeforeEach donc disponible partout=**

**3ème: je fais mon assertion en utilisant la librairie Hamcrest : ici je vérifie que mon absence lundiJeViensPasJaiPiscine a bien été ajoutée à la collection d’absences associée au Salarie (et accessible via l’accesseur correspondant, ici getListeAbsences()**

1. **Fournir le code de la méthode *refuser* de la classe** DemandeAbsence**.**

****

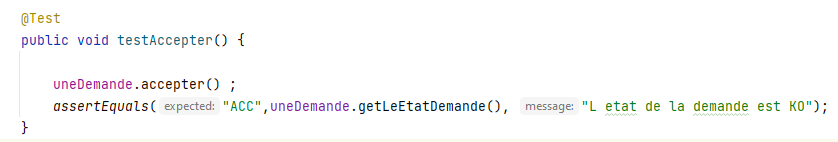
**Erreurs à éviter :**

**Oublier de répondre à une partie de la question (Etat de la demande dans Absence, ou motif)**

**Question n°4 : préliminaire. Bonus pour le modèle pas bon (testAccepter()). On le voit car**

1. **Le test ne passe pas dès le départ alors que la méthode n’est pas à modifier**
2. **Les « hints » ne correspondent pas aux paramètres …**

**C’est une question bonus.**



1. **Dans la classe de test unitaire** TestDemandeAbsence**, ajoutez une méthode de test de la méthode *refuser* de la classe** DemandeAbsence***. On vérifiera que le motif et l’état de l’absence ont bien été affectés.***

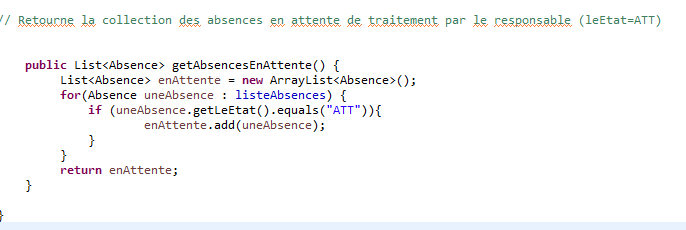
****

**Erreurs à éviter :**

**Oublier de répondre à une partie de la question (test du motif)**

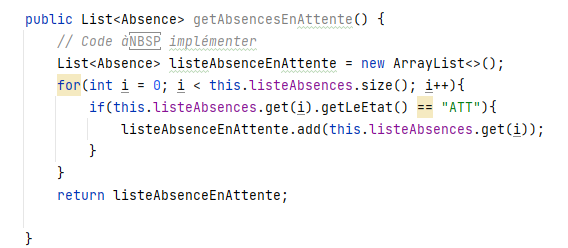
**Partir sur un modèle en partie défaillant (testAccepter() ) : en étude de cas on respecte le modèle, dans la réalité et la pratique on écrit du code performant qui fonctionne**

1. **Fournir le code de la méthode** *getAbsencesEnAttente* **de la classe** Salarie**.**

****

**Dans l’EDC il ne s’agissait même pas de la question la plus difficile (que j’ai enlevé). Cf méthode getJoursAbsence de la classe Absence à coder dans l’EDC initiale.**

**Autre code beurk mais OK**

****

**Méthode : on trouve toujours une ou plusieurs méthodes de ce genre à implémenter en EDC, avec les phases suivantes.**

**Objectif : retourner une collection d’objets selon une ou plusieurs conditions**

**Etapes :**

1. **Déclaration et instanciation de la collection ) retourner**
   1. **Déclaration, partie gauche, il faut la bonne interface (List le plus souvent), le bon type d’objet (ici Absence) et un nom qui montrer l’intelligence remarquable de l’apprenant (ici listeAbsencesEnAttente)**
   2. **Instanciation, partie droite : new avec la « bonne » classe concrètre (ArrayList le plus souvent). La « magie » de java rend inutile le typage ici (entre les <>**
2. **Une boucle**
   1. **De préférence de type foreach (attention la boucle foreach en java s’écrit également for)**
   2. **Bonne SYNTAXE : pour le foreach java : for (Classe objetBoucle : collection)**
   3. **Attention aux particularités de la boucle choisie (borne sup pour le for classique, incrémentation pour while …)**
3. **Une condition**
   1. **Syntaxe : if (condition) où condition renvoie un boolean (type primitif)**
   2. **Attention on réserve le == à la comparaison avec un type primitif (boolean, entier) sinon on préfère la méthode .equals de l’objet considéré**
4. **Une ou plusieurs instructions** 
   1. **Au moins une correspondante au cas où la condition est vraie**
   2. **Eventuellement d’autres si un else est attendu**
5. **Un return si la méthode à implémenter est une *fonction* (pas de return si c’est une procédure)**
6. **Dans la classe de test unitaire** TestSalarie**, ajoutez une méthode de test de la méthode *getAbsencesEnAttente* de la classe** Salarie

****

**Remarques :**

**1.d’autres réponses étaient acceptées.**

**2. La réponse proposée impliquait soit (a ou b)**

**a. de corriger un manque dans le constructeur de Absence**

this.leEtat = "ATT";

**b. de positionner explicitement l’état de la demande dans le test unitaire pour chaque Absence, avant le test assertThat**

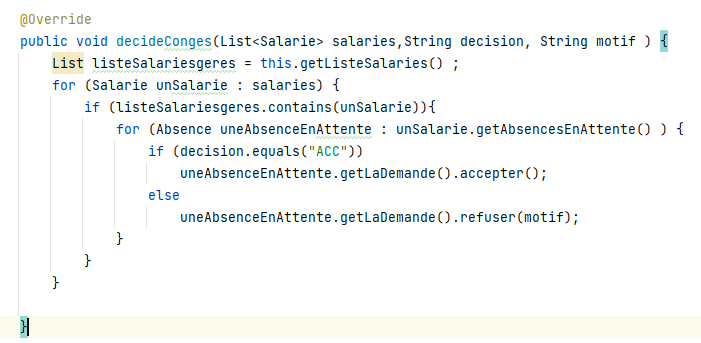
****

**…**

****

**Ici la solution correspondant à la correction a est présentée**

1. **Répondez à l’évolution du besoin décrite dans le document 5**

****

**Remarques : pas de grosse complexité technique il faut bien comprendre le besoin et s’appuyer sur les questions précédentes pour faire au plus simple.**

1. **Ecrire un test unitaire de la méthode *decideConges* de la classe** Manager

**Un exemple très complet**

**D’autres solutions sont possibles évidemment**