TP2 : Test unitaire de l'application *Pigeon*

Objectifs

- Concevoir et implémenter une application orientée objet Java simple
- Implémenter et exécuter un ensemble de tests unitaires pour cette application en utilisant JUnit
- Mettre en œuvre l'écriture de tests unitaires par classes d'équivalence et par analyse des valeurs limites

Recommendations pour l'écriture des tests unitaires

- Convention des trois A : séparez bien les parties initialisation (*Arrange*), interaction (*Act*) et validation (*Assert*), lorsque c'est possible [a]
- Utilisez des *fixtures* pour regrouper le contexte commun à plusieurs tests dès que nécessaire [b]
- Selectionnez les assertions les plus appropriées et spécialisées pour l'écriture de vos oracles [c]
- Utilisez des noms clairs et explicites pour vos cas de test , par exemple `divide_shouldThrow_whenZero()` ; lorsque nécessaire, associez-leur un label personnalisé [d]
- Certains aspects de la spécification sont volontairement sous-spécifiés ; faites vos propres choix mais assurez-vous que vos tests les vérifient

a. Parfois, par exemple pour tester la levée d'une exception, il est plus difficile de séparer les parties *Act* et *Assert*
b. https://junit.org/junit5/docs/current/user-guide/#writing-tests-classes-and-methods
c. https://junit.org/junit5/docs/5.0.1/api/org/junit/jupiter/api/Assertions.html
d. https://junit.org/junit5/docs/5.0.3/api/org/junit/jupiter/api/DisplayName.html

1 L'application *Pigeon*

Nous nous proposons de développer les bases d'une application nommée *Pigeon*. [1] L'objectif principal est de développer les classes métiers au cœur de cette application, et de les tester unitairement à l'aide du *framework* JUnit. Nous verrons plus tard comment enrichir cette application (API REST, base de données), mais nous nous concentrons ici sur le test unitaire des classes métiers les plus importantes.

Pigeon est une application de microblogging qui permet à ses utilisateurs de partager des messages courts (appelés *chirps*). Chaque utilisateur possède une *timeline* où lui est présentée une liste personnalisée de *chirps* en ordre antéchronologique. Un mécanisme d'abonnement permet aux utilisateurs de s'abonner à d'autres utilisateurs. Si un utilisateur X s'abonne à l'utilisateur Y, alors X voit apparaitre les *chirps* rédigés par Y dans sa *timeline*. Un utilisateur peut "aimer" un *chirp* (ou annuler cette action), et chaque *chirp* comporte un compteur du nombre de fois où il a été aimé, et par quels utilisateurs.

Enfin, un utilisateur peut décider de *re-publier* n'importe quel *chirp* (ou d'annuler sa re-publication). Si un utilisateur re-publie un *chirp*, alors tous les abonnés de cet utilisateur le verront apparaître sur leur *timeline*, avec une mention précisant quel est l'utilisateur à l'origine de la re-publication.

2 La classe `User`

Dans *Pigeon*, un utilisateur est caractérisé par son identifiant d'utilisateur (entre 4 et 24 caractères, unique pour chaque utilisateur, sans espace), son nom d'usage (entre 4 et 24 caractères, qui peut ne pas être unique)

1. Qui ne saurait ressembler à un réseau social populaire maintenant disparu.

et sa date d'inscription.[2] Un utilisateur possède en outre une liste des utilisateurs auxquels il s'est abonné, et un utilisateur peut s'abonner ou se désabonner d'un autre utilisateur (par exemple à l'aide de méthodes `subscribe(User)` et `unsubscribe(User)`). Enfin, un utilisateur peut en bloquer un autre, auquel cas il ne voit pas apparaître ses *chirps* sur sa *timeline*, même s'ils s'ont re-publiés par l'un de ses abonnements.

- Implémentez la classe `User`, dont les instances représentent des utilisateurs de *Pigeon*, en suivant les bonnes pratiques de programmation orientée objet ; ne vous préoccupez pas du code d'implémentation de ses méthodes pour l'instant, mais seulement de son interface.
- Implémentez ensuite un ensemble de cas de test pour la classe `User`, agrégés dans sa classe de test dédiée `UserTest`. Réfléchissez aux vecteurs de test appropriés, en veillant à valider à la fois les cas *valides* et les cas *invalides*. En guise d'inspiration, voici quelques exemples :
  - Que doit-il se passer si la date d'inscription de l'utilisateur se situe dans le futur ? Une solution possible est de lever une exception à la création d'un tel utilisateur (spécifique, que vous définirez vous-même ou bien générique, comme `IllegalArgumentException`). Vérifiez-le dans vos tests.
  - Un utilisateur ne peut évidemment ni s'abonner à lui-même, ni s'abonner plusieurs fois à la même personne. Vérifiez-le dans vos tests.
  - Lorsqu'un utilisateur s'abonne à un autre utilisateur, alors cet utilisateur doit apparaître dans la liste de ses abonnements.
  - Est-ce la responsabilité de la classe `User` de vérifier que les identifiants utilisateurs sont uniques ? Pourquoi ?
- Terminez l'implémentation de la classe `User` de sorte à ce que tous vos tests passent avec succès.

3 La classe `Chirp`

Dans *Pigeon*, un *chirp* est un message non-vide de 80 caractères au maximum qui possède un identifiant unique, un auteur, une date et une heure de publication. Un utilisateur peut aimer un *chirp* ou le re-publier (par exemple, à l'aide de méthodes `like(User)` et `broadcast(User)`), et les *chirps* gardent trace de ces activités (par exemple, combien de re-publications, et par quels utilisateurs). On considère que, initialement, chaque *chirp* est aimé par son auteur.

- Implémentez la classe `Chirp` en suivant les bonnes pratiques de programmation orientée objet ; ne vous préoccupez pas du code d'implémentation de ses méthodes pour l'instant, mais seulement de son interface.
- Implémentez ensuite un ensemble de cas de test pour la classe `Chirp`, agrégés dans sa classe de test dédiée `ChirpTest`. Réfléchissez aux vecteurs de test appropriés, en veillant à valider à la fois les cas *valides* et les cas *invalides*. En guise d'inspiration, voici quelques exemples (certains des exemples discutés pour la classe `User` s'appliquent également ici) :
  - Que doit-il se passer si un utilisateur essaye de re-publier l'un de ses propres *chirps* ? Faites un choix et vérifiez-le dans vos tests.
  - Que doit-il se passer si un utilisateur essaye d'aimer ou de re-publier plusieurs fois le même *chirp* ? Faites un choix et vérifiez-le dans vos tests.
  - Il n'est pas possible qu'un *chirp* ait été aimé moins d'une fois (l'auteur aime toujours ses propres *chirps*) ou re-publié un nombre négatif de fois.
- Terminez l'implémentation de la classe `Chirp` de sorte à ce que tous vos tests passent avec succès.

4 La classe `Timeline`

La classe `Timeline` présente des comportements plus complexes que les classes `User` et `Chirp`. On considère ici qu'une *timeline* est liée à un utilisateur en particulier.

2. Vous pourrez par exemple utiliser la classe LocalDateTime du JDK pour représenter les dates.

- Réfléchissez à l'interface de cette classe d'après les descriptions ci-dessus. Quels services doit-elle offrir ? Comment interagit-elle avec les classes `User` et `Chirp` ? Écrivez une première version de cette interface en incluant (parmi d'autres) les méthodes suivantes :
  - Une méthode doit permettre de récupérer un nombre configurable de *chirps*, en ordre antéchronologique et antérieurs à une date passée en paramètre afin de permettre la pagination de la *timeline*
  - Une méthode doit permettre de rechercher des *chirps* par mots-clés ; une autre, par auteur(s)
  - Une méthode doit permettre de retourner les *chirps* d'une *timeline* par ordre décroissant de "j'aime" et/ou de re-publications
- Proposez un ensemble de cas de test pour cette interface et implémentez-les. Laissez libre cours à votre imagination lorsque les exigences sont lâches, mais assurez-vous que vos tests les vérifient et que leurs noms permettent de comprendre les choix qui ont été faits. En guide d'inspiration, voici quelques exemples :
  - Vérifiez que les *chirps* apparaissent bien en ordre antéchronologique dans la *timeline*
  - Un utilisateur qui ne suit personne ne doit voir que ses propres *chirps*
  - Si un utilisateur est bloqué, ses *chirps* ne doivent pas apparaître dans la *timeline*, même s'ils ont été re-publiés par d'autres ou si une recherche par mots-clés ou auteur devrait les faire apparaître
  - Lorsqu'un utilisateur est bloqué, ses *chirps* disparaissent instantanément de la *timeline*
  - À la suite d'un abonnement, les *chirps* de l'utilisateur auquel l'on s'est abonné doivent apparaître immédiatement dans la *timeline*. Inversement, après un désabonnement, ils disparaissent instantanément

Rendu

À l'issue de la séance, archivez le projet contenant le code de l'application ainsi que ses tests et partagez-les à l'adresse `thomas.degueule@labri.fr` avec le sujet **I3GLTest – Nom1(-Nom2)**. Les cas de test sont plus importants que le code de l'application : *il n'est pas nécessaire de s'assurer que tous les tests écrits s'exécutent avec succès.* Assurez-vous cependant que la commande `mvn test` permet de lancer l'exécution de tous vos tests. Ce rendu n'est pas noté et ne servira qu'à vous faire des retours et mieux orienter le reste du module.