Killian Le Page
Christopher Pico

Thaw - Architecture du projet

[INSÉRER LOGO DANS CETTE PAGE SI ON EN A]

Table des matières :

I - Architecture par package

1. Règles de nommage

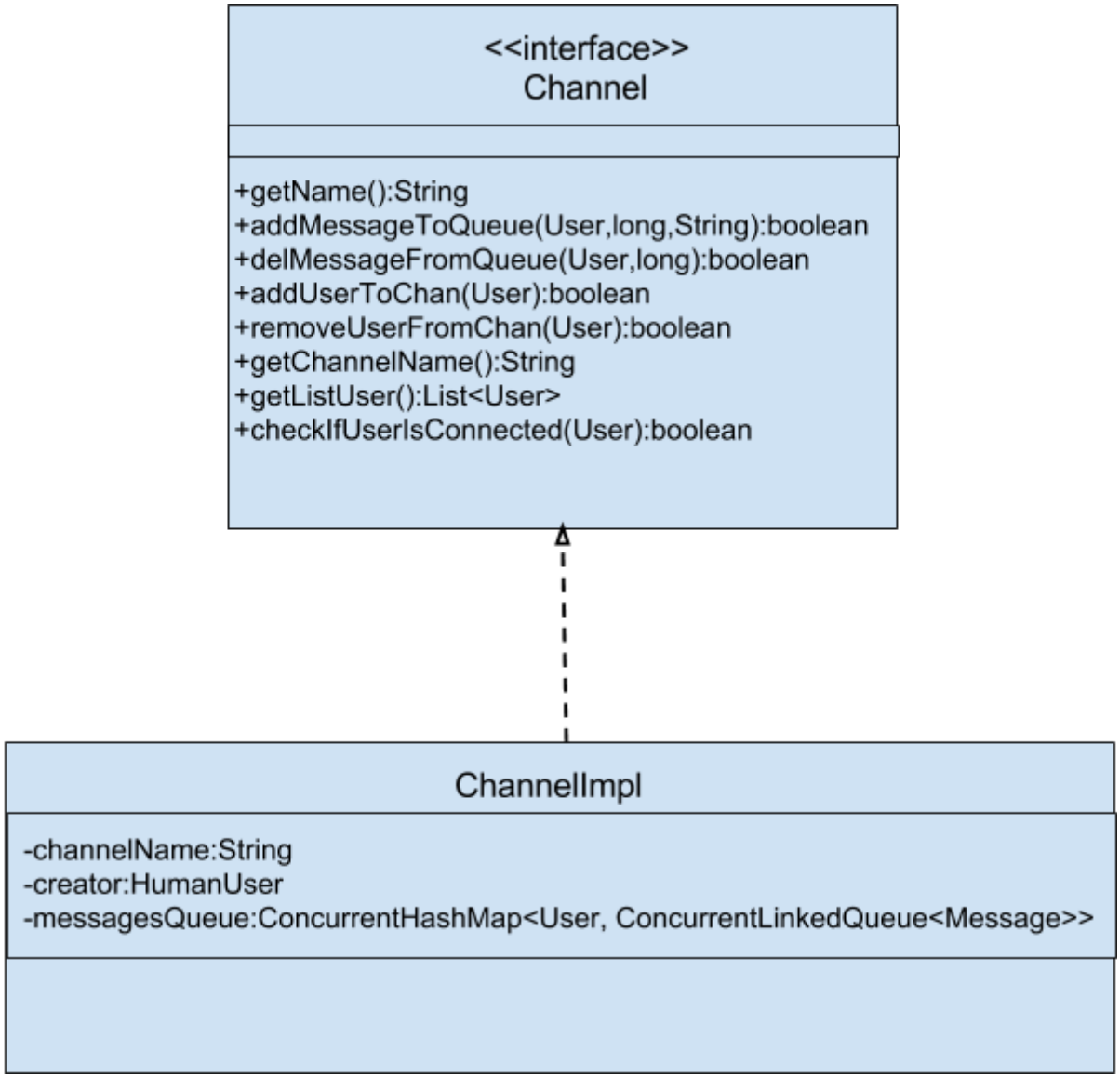
Afin d'harmoniser le travail, il nous a fallu trouver une règle de nommage pour les différentes classes, interfaces et package.

Nous sommes arrivés aux règles suivantes :

- Un package devra correspondre à un rôle clef (représentation d'un channel, un user, programme principal, etc..).
- Les noms des méthodes, classes, interfaces et les commentaires/javadoc seront exclusivement en anglais.
- Les interfaces seront nommées en fonctions de ce qu'elle représente. Ainsi, l'interface représentant un canal de discussion se nommera Channel, celle pour les utilisateurs User etc…
- Les classes implémentant ces interfaces se nommeront comme ces dernières mais en finissant avec Impl (pour implémentation) : par exemple ChannelImp.
  Si les interfaces peuvent représenter un ensemble, alors
  nous feront une classe pour chaque distinction possible.
  Par exemple : HumanUser ou bien encore Bot.
- Les rares classes abstraites se nommeront de cette manière:
  AbstractMonInterface / AbstractMaClasse
- Enfin, si une classe ne possède pas d'interface, il ne sera pas nécessaire de mettre Impl à la fin de cette dernière (cf les classes Message et Server)

2. Diagrammes UML

A) Channel



Nous avions souhaité représenter un canal de discussion par 3 champs principaux : Son nom, le nom du créateur ainsi qu'une file de message différente entre chaque canal.

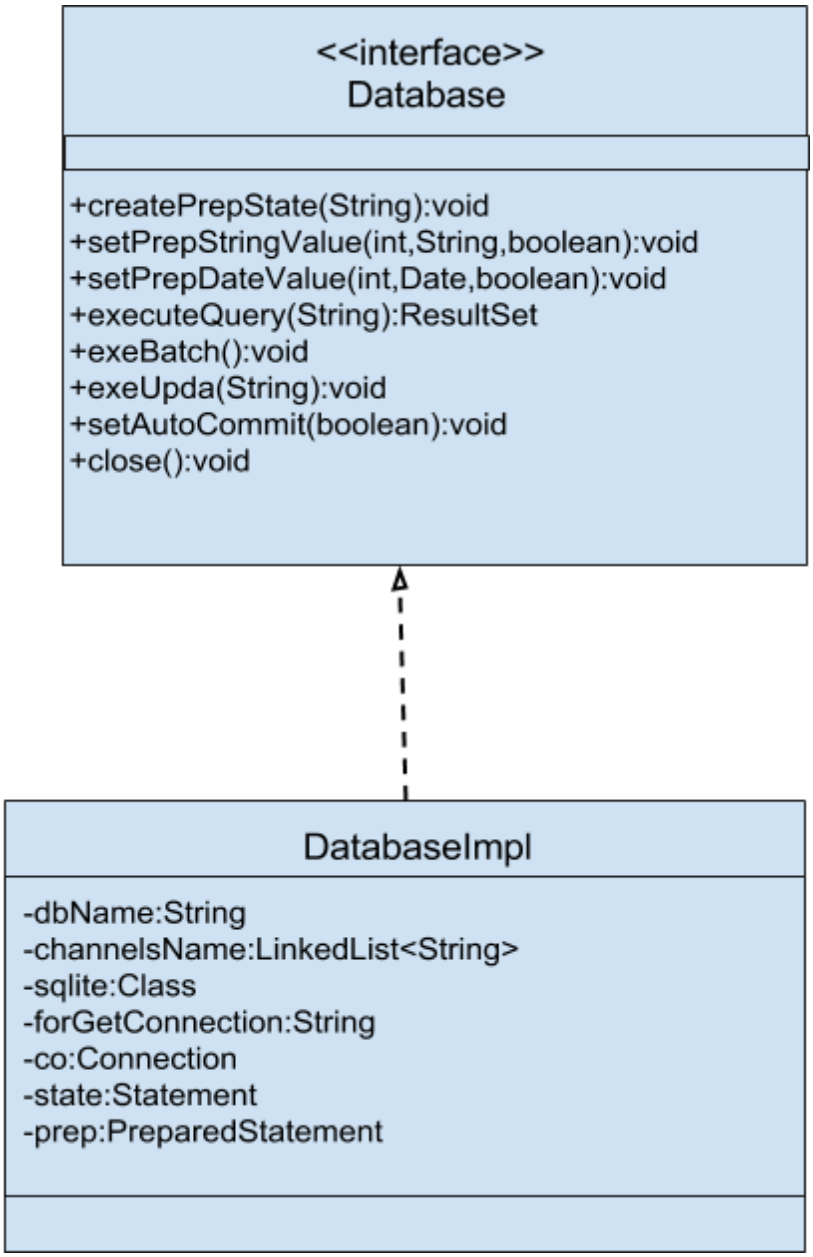
Comme cette file de message pourrait être consultés par plusieurs processus, il nous faut garantir un accès concurrent sans problème afin d'éviter des problèmes lors de la lecture/écriture des différents messages.

Le choix de l'interface nous permettait ici de concentrer la javadoc en un endroit mais surtout de pouvoir cacher l'implémentation de nos différents champs à l'extérieur.

B) configuration

TODO:voir si classe finie + finir diagramme UML

C) database



Afin de simplifier la manipulation de notre base de donnée, nous avons décidé d'implémenter l'interface Database et la classe implémentant : DatabaseImpl.

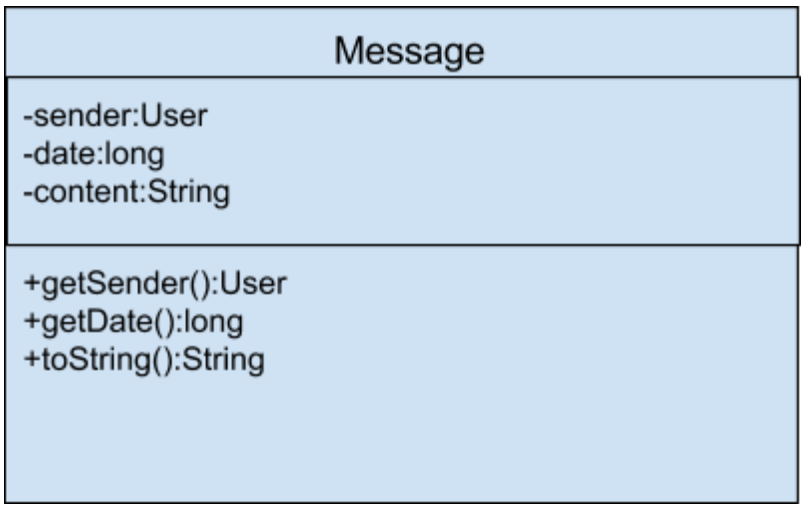
Les différents champs de classe DatabaseImpl représentent ce que nous devons manipuler pendant l'existence de la base de données afin de manipuler cette dernière. Nous ne devons lancer pas plus d'une instance de cette dernière lors du lancement du serveur puisque tout les traitement seront centralisés dans cette dernière.

L'interface permet de centraliser encore une fois la javadoc mais surtout de préciser les différentes méthodes que la classe devra implémenter au fur et à mesures de besoin. Elle permet aussi de se limiter dans le choix des méthodes que nous pouvons utiliser et de permettre l'automatisation des différents traitement à venir.

D) main

TODO : diagramme UML + Explication

E) message

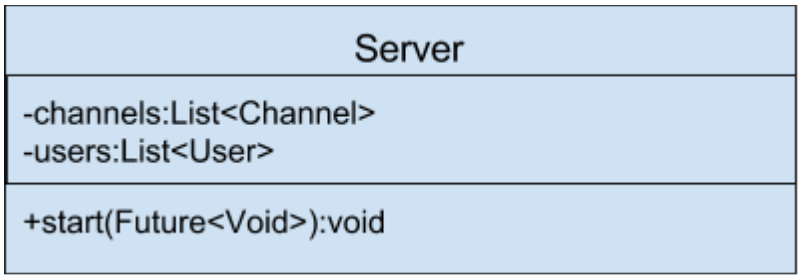


Afin d'unifier la façon dont nous traitons ls messages, nous avons créé cette classe.

Les champs présents ici permettent de représenter plus facilement la manière dont nous allons stocker un message dans notre base de données.

Puisque les messages n'ont d'autres vocations que d'être stockés ou supprimer, nous n'avons pas eu besoin d'interface ou bien encore de fonction de manipulation spécifique.

F) Server

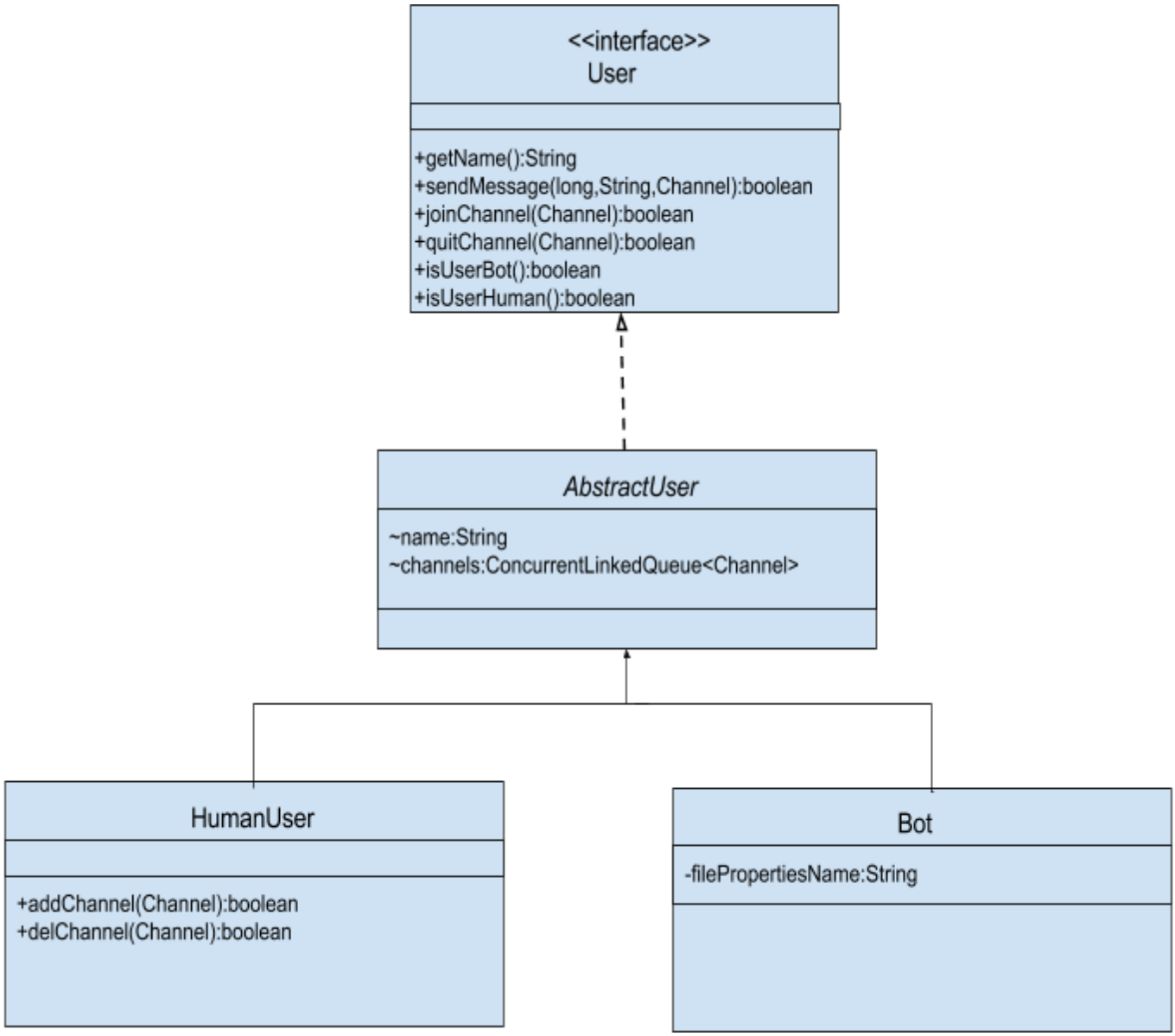


Cette classe permet de représenter les différents actions que pourra faire notre serveur.

Les champs symbolisent l'ensemble des utilisateurs et canaux présent dans la base de donnée.

De par le fait que nous devons contrôler au mieux les actions faites par le serveur, nous n'avons qu'une unique méthode public permettant de démarrer le serveur. Les traitements étant figés, il n'y a aucun besoin de rendre plus de méthode visible depuis l'extérieur de l'objet.

G) user



Ce package permet de regrouper toutes les méthodes et abstractions nécessaire à un utilisateur qu'il soit humain ou bot.

De ce fait, il nous fallait d'abord définir une interface permettant de regrouper toutes les méthodes nécessaires à ces utilisateurs. Ensuite, puisque bot et humain partage certains très cohérent et utiles en commun (tel qu'un nom ou la liste des channels dans lequel cet utilisateur est présent), la création d'une classe abstraite AbstractUser paraissait cohérente et très utile. Enfin, il nous ne restait plus qu'à définir les méthodes propres aux bots et utilisateurs humains dans une classe qui leur était propre à chacun.

En procédant ainsi, nous avons réussi à isoler au mieux ces classes et faciliter l'écriture de ces dernières en facilitant la réutilisation de code.

II - Architecture globale

En excluant le main, nous arrivons donc à la représentation suivante :

TODO

[ICI CE SERA UN GROS DIAGRAMME]