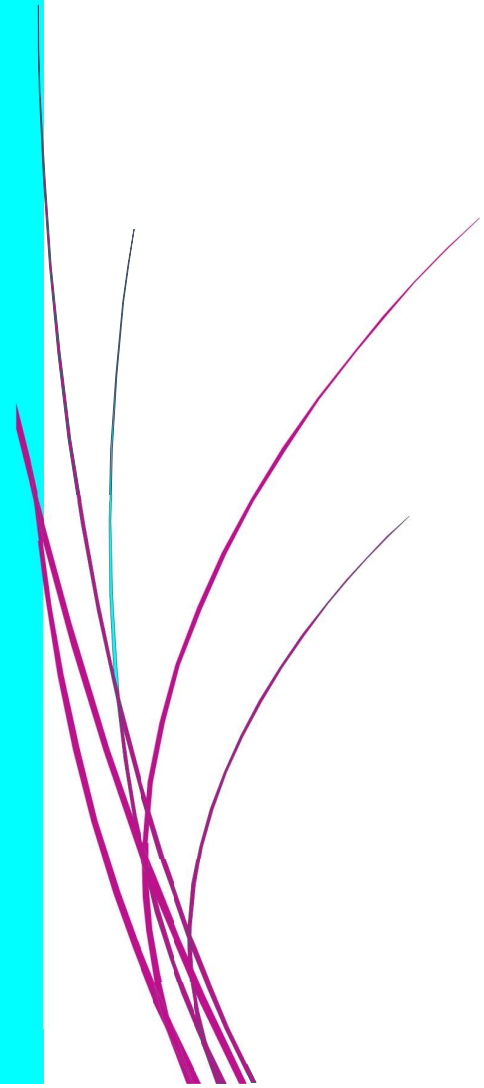


14/04/2022

Rapport Laboratoire 2 : Squadron



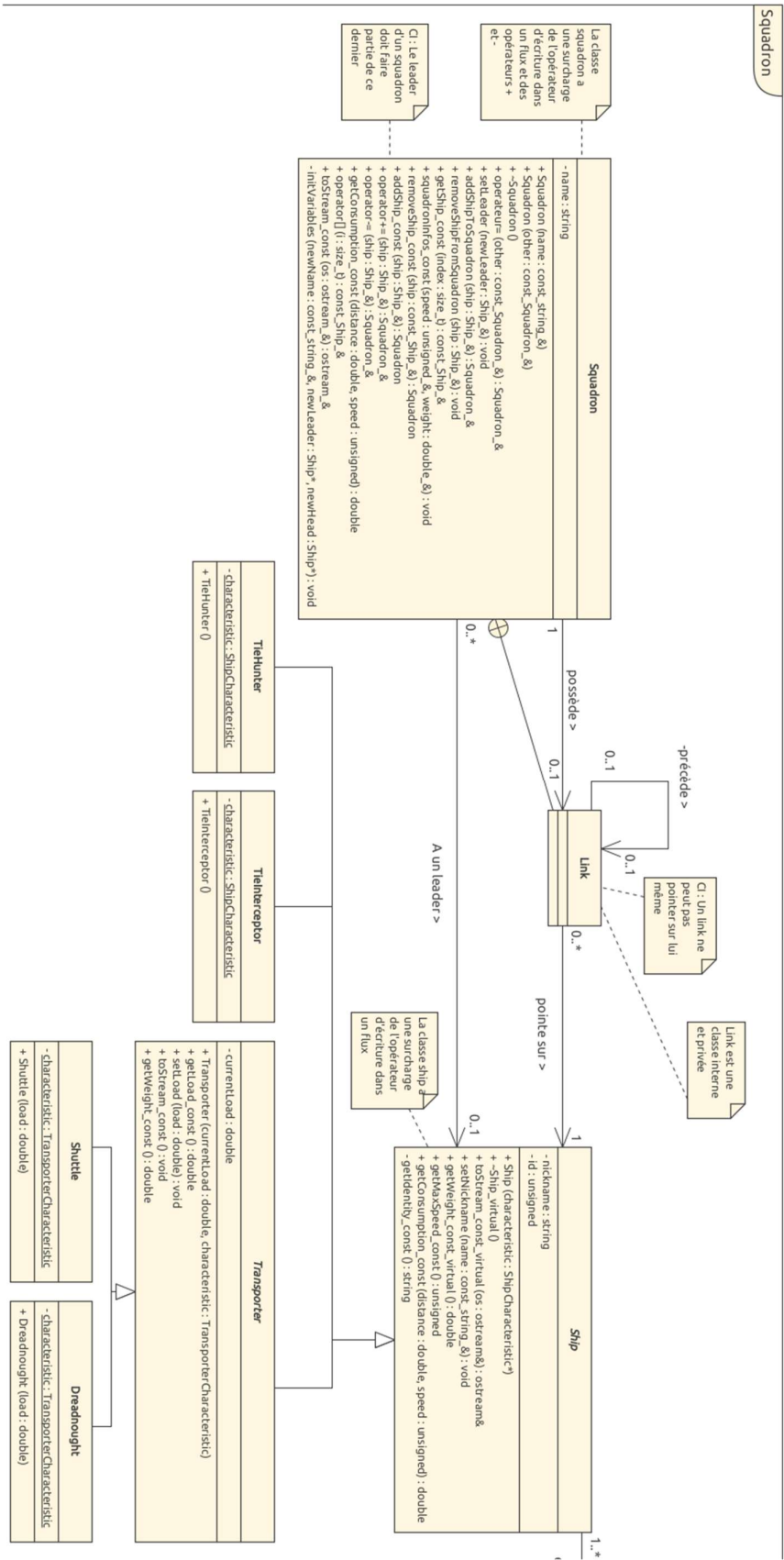
Alexandre Jaquier et Jonathan Friedli

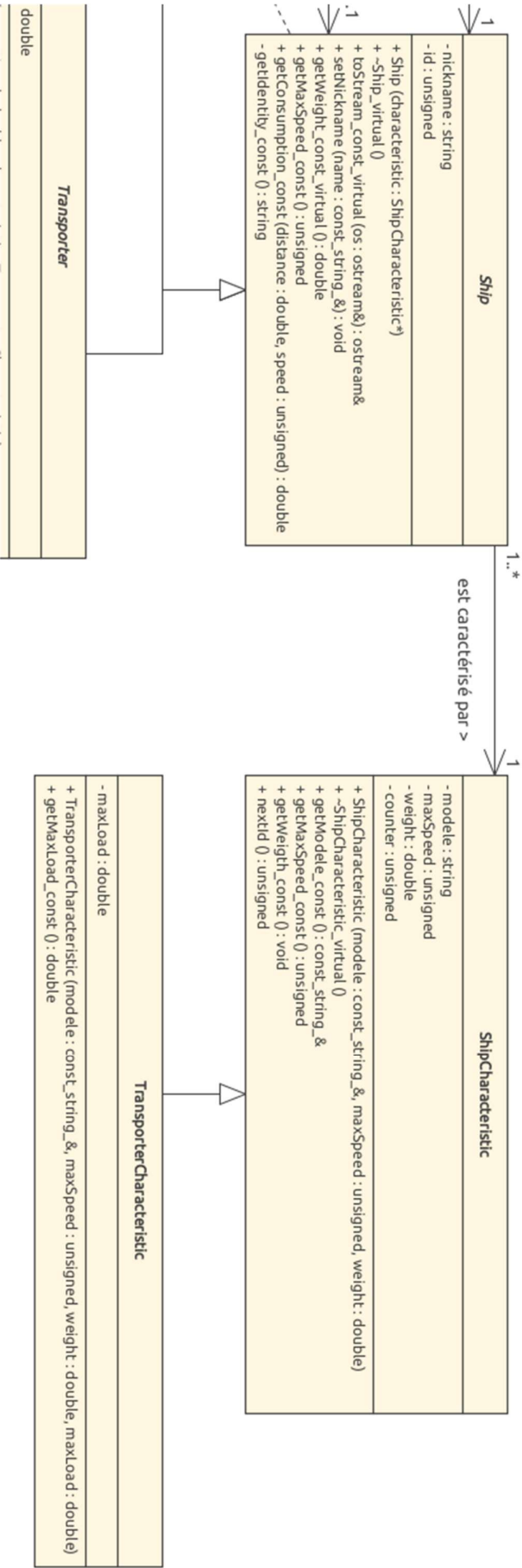
Introduction

Dans le cadre de ce laboratoire nous devons implémenter une classe Squadron permettant de gérer une escadrille de vaisseaux spatiaux. Pour ce faire, nous devons également implémenter certaines classes qui modélisent des vaisseaux spatiaux. Il y a 2 types de vaisseau différents, les vaisseaux « simple » et les vaisseaux pouvant transporter des cargaisons.

Chaque escadrille (« Squadron ») a un nom et peut ajouter et supprimer des vaisseaux. Une escadrille peut également avoir un leader. Ce leader doit faire partie de l'escadrille avant d'être promu. Si le leader est démis de ses fonctions, il continue de faire partie de l'escadrille. Une escadrille ne peut se déplacer qu'à une vitesse pouvant être atteint par tous les vaisseaux qui font partie de ladite escadrille. Nous pouvons également calculer le carburant que va consommer cette escadrille en fonction de la distance et de sa vitesse. Il doit être possible d'afficher les détails d'une escadrille (infos de l'escadrille et les vaisseaux qui la composent).

Diagramme UML





Choix de modélisation et d'implémentation

Vaisseau

Comme montré dans l'UML, nous avons fait sur un héritage afin de factoriser le code des vaisseaux. Les vaisseaux simples (« TieHunter » et « TielInterceptor ») ont une vitesse maximale, un poids, un modèle et un identifiant. Les vaisseaux transportant des cargaisons (« Shuttle » et « Dreadnought ») ont, en plus des informations citées précédemment, un poids maximal pour la cargaison et le poids de la cargaison qu'il transportent actuellement.

Toutes ces informations étant propre au type de vaisseau, nous avons d'abord pensé les stocker sous forme de constantes statiques. Cependant cela nous posait quelques problèmes de factorisation, notamment au niveau du poids de la cargaison. Nous avons donc fait le choix, de créer une classe « ShipCharacteristic » et respectivement « TransporterCharacteristic » afin de stocker ces valeurs. Nous créons ensuite dans les vaisseaux (« TieHuter », « Shuttle ») des variables statiques de type « ShipCharacteristic » ou « TransporterCharacteristic ». Dans « Transporter », nous stockons un pointeur sur le « TransporterCharacteristic » et dans Ship, un pointeur sur le « ShipCharacteristic ».

Squadron

Afin de pouvoir gérer les vaisseaux faisant partie de l'escadrille, nous avons créé notre propre structure de données. Nous avons décidé de créer notre variante de la liste chaînée, avec une struct « Link » représentant un vaisseau. « Link » est une classe interne à Squadron et est privée. Chaque « Link » a un pointeur sur « Ship » et un pointeur sur le prochain « Maillon ».

Afin d'ajouter et de supprimer des vaisseaux de l'escadrille, nous avons surchargé les opérateurs « + », « += », « - » et « -= ».

Afin de pouvoir récupérer le $n^{\text{ème}}$ élément du squadron nous avons surchargé l'opérateur []. Afin de récupérer cet élément, nous devons parcourir notre liste. Cette méthode est donc en $O(n)$. Il est important de noter que nous ne pouvons pas faire d'affectation via l'opérateur []. (Exemple : il n'est pas possible de faire « `squad[4] = tie3 ;` »).

Afin de calculer la consommation d'un squadron en fonction de la vitesse et de la distance, nous calculons la consommation de chaque vaisseau. Si la vitesse est trop élevée pour le vaisseau, nous lançons une exception.

Tests effectués

| Test | Résultat attendu |
|---|------------------|
| Il est possible de créer des vaisseaux simples (TieHunter et TieInterceptor). Les id sont générés automatiquement. | OK |
| Il est possible de créer des vaisseaux avec une cargaison (Shuttle et Dreadnought). Les id sont également générés automatiquement. Si le poids de la cargaison est trop grand, une erreur se lance. | OK |
| Il est impossible d'utiliser une méthode pas constante sur un vaisseau constant. (Exemple : Si on a un shuttle constant, on ne peut pas utiliser setLoad()). | OK |
| Il est possible de renommer un vaisseau via la méthode setNickname. | OK |
| Il est possible d'afficher un vaisseau grâce à la surcharge de l'opérateur d'écriture dans un flux. | OK |
| Il est possible de créer un squadron et d'y ajouter/supprimer des vaisseaux. | OK |
| Il est également possible de créer un squadron via le constructeur de copie. | OK |
| Il est possible d'afficher un squadron grâce à la surcharge de l'opérateur d'écriture dans un flux. | OK |
| Il est possible de calculer la consommation d'un squadron pour une distance et une vitesse donnée. Si la vitesse est trop élevée, une exception est lancée. | OK |
| Les opérateurs +, - sont bien surchargés pour la classe squadron et peuvent être utilisés sur un squadron constant. | OK |
| Les opérateurs += et -= sont bien surchargés pour la classe squadron mais ne peuvent pas être utilisés sur un squadron constant. | OK |
| Il est possible d'affecter un squadron à un autre via l'opérateur d'affectation. | OK |
| Il est possible de retourner le n ^{ème} membre du squadron via la surcharge de l'opérateur []. Si n est trop grand, une erreur se lance. | OK |