TP C++ : Fourmis

Les Fourmis sont des petits bêtes très communicantes, qui ont pour objectif de créer une colonie.
Les fourmis vont puiser de l'énergie dans leur environnement, puis faire naitre une reine, qui fera naître d'autres fourmis
Le cycle de vie classique de ces fourmis est :

- Je cherche des objets,
- J'absorbe leur énergie,
- Je communique avec les autres fourmis pour augmenter de catégorie,
- Je crée une Reine quand je le peux,
- La Reine crée de nouveaux fourmis,
- etc...

Nous allons suivre dans ce TP la vie de deux petits fourmis. : 327e et 56e.

<u>Commençons par créer nos fourmis.</u>

Chaque fourmi est représenté par :

- Une catégorie (un entier)
- Un identifiant (chaine)
- Un niveau d'énergie (un entier)

Chaque fourmi peut :

- Se charger (augmenter son énergie)
- Agir (diminuer son énergie)

1. Créez la classe Fourmi.

Fourmi
categorie id energie
charger() agir()

★ Attention à faire les bonnes inclusions, les bonnes portées, et à bien séparer vos fichiers.

2. Créez une méthode ping()

❑ Cette méthode permettra à chaque fourmi d'afficher son identifiant, sa catégorie, et son énergie.

Fourmi
categorie id energie
charger() agir() ping()

3. Surchargez le constructeur

❑ Ceci permettra de pouvoir choisir la valeur des attributs lors de la création d'une fourmi.

★ Par défaut, les fourmis sont de catégorie 1, ont 100 d'énergie, et un identifiant "Reese".

Bon, assez codé, construisons nos deux vedettes.
Instanciez deux fourmis : 56e, et 327e.

★ Attention, passez les bons paramètres à votre constructeur pour créer vos deux fourmis.

❖ Vérifiez par des pings que chacune est bien qui elle doit être.

4. Codez les méthodes charger et agir.

❑ Celles-ci agissent sur l'énergie de la fourmi (respectivement en l'augmentant et en la diminuant).

C'est bien, mais si il suffisait qu'elles se chargent toute seules dans leur coin, 56e et 327e auraient la vie facile. En réalité, elles sont obligées de consommer des choses pour en tirer de l'énergie.

5. Créons donc une classe Chose

❑ Celle-ci ne sera représentée que par un niveau d'énergie (un entier).

Chose
energie

Ok, l'idée maintenant est de faire en sorte que l'énergie de nos fourmis n'augmente qu'en consommant des Choses.

6. Modifions la méthode charger.

- ❏ Renommons la en absorber(), et faisons en sorte qu'elle prenne en paramètre une chose. C'est l'énergie de cette chose qui sera absorbée.

Fourmi
categorie id energie
absorber() agir() ping()

Problème : l'énergie d'une chose est privée. Or, la seule raison d'être d'une Chose est d'avoir une énergie à disposition. Rajoutons donc une méthode getEnergie() publique, qui nous renverra l'énergie de chaque Chose.

- ❏ Créez une chose(énergie au choix) et vérifiez que 56e et 327e peuvent l'absorber.

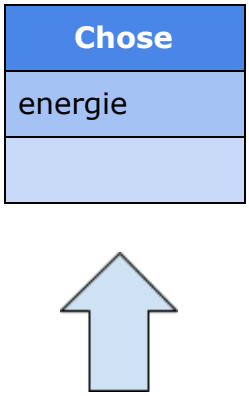
- ★ Oui mais, dans notre cas, un objet absorbé peut toujours l'être. Modifions un peu notre code pour qu'un objet absorbé ne soit plus disponible. Comment? Eh bien réduisons à 0 l'énergie d'un objet quand il est absorbé. (Changeons la méthode getEnergie en syphoner(). Syphoner renverra l'énergie de la chose, et mettra cette énergie à 0.). Attention, la chose existe toujours, mais elle n'a plus d'énergie.

Exemple :
Plante (én.150) est absorbée par 56e (én.100). Plante a maintenant 0 en énergie, et 56e 250.

❖ Testez votre code avec quelques appels à ces méthodes.

Ok, on peut maintenant se nourrir de choses.

7. Un fourmi est une chose !



★ Attention, plus complexe : On peut considérer que les fourmis sont des choses. Une fourmi doit donc pouvoir absorber une autre fourmi.

❑ Ce qu'il faut se dire, c'est qu'une fourmi est un type particulier de chose, et peut donc hériter de Chose.

★ Attention ! Il va falloir adapter les constructeurs, les portées, et les attributs en conséquence ! (recherchez "liste d'initialisation" pour vous aider avec les constructeurs)

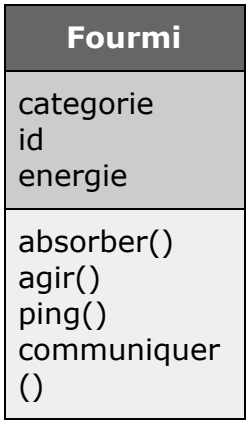
Après cette étape, 56e devrait pouvoir absorber 327e.

Mais bon, les fourmis ont cet avantage de réfléchir au bien commun avant le leur, elles vont donc se focaliser sur d'autres choses plutôt que sur des fourmis.

Nos fourmis peuvent donc se nourrir. Elles vont maintenant devoir apprendre les unes des autres pour pouvoir évoluer.

8. Codons la méthode communiquer

❑ Cette méthode permet à un fourmi de monter de catégorie en communiquant avec une autre fourmi.



★ Attention, il y a certaines règles à suivre:

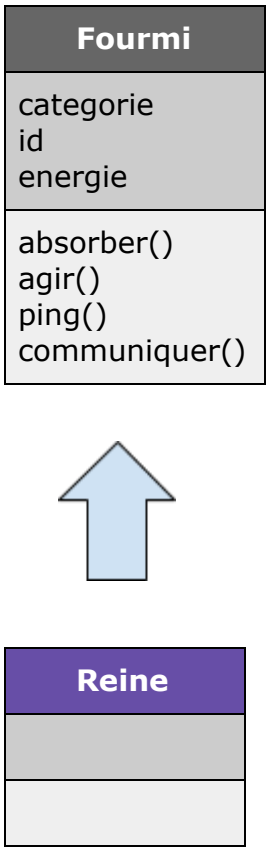
★ Communiquer coûte de l'énergie. Le fourmi qui communique doit dépenser la moitié de l'énergie de sa cible * sa catégorie.
Exemple : Si 327e (cat 3 energie 200) veut absorber 56e (cat 2 energie 100), elle devra dépenser 2*50=100 énergie. 327e passera donc en catégorie 4, et aura 100 d'énergie restante).

★ Communiquer ne modifie pas la fourmi ciblée.

❏ N'oublions pas de tester !

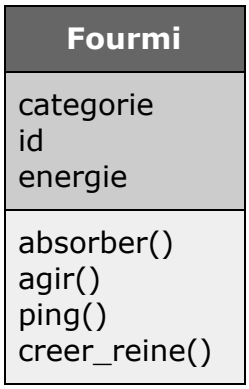
Nos fourmis ne sont toujours pas en mesure de se répliquer…. Il est temps d'y remédier.
Lorsqu'une fourmi a accumulé assez de connaissances, elle peut construire une Reine, qui sera en charge de créer de multiples fourmis.

9. Construisons notre classe Reine



★ Attention ! Une Reine est bien sûr un fourmi ! (conseil : forward dec dans Fourmi.h de la classe Reine, et un include dans Fourmi.cpp)

10. Permettons à une fourmi de pouvoir créer une Reine.



★ Attention ! Seule une fourmi de catégorie 5 ou plus qui n'a pas de reine peut construire une Reine.

★ De plus, les Fourmis ne peuvent avoir qu'une seule Reine. Il va donc falloir que chaque fourmi sache qui est sa reine. Par défaut, les fourmis n'ont pas de reine (56e et 327e n'en ont donc pas. Celà dit, une fois qu'une fourmi crée une reine, elle devient sa reine, et chaque fourmi créé par cette reine lui sera affiliée. Il faut donc rajouter à chaque fourmi cette information. (Un pointeur serait le bienvenu).

★ Enfin, créer une reine est coûteux en énergie (1000/categorie). Vérifiez bien qu'une fourmi peut créer sa reine avant de la créer….

11. Ajoutons un constructeur à notre reine

❏ Ceci permettra que son id contienne celle de son créateur (une reine créée par 327e s'appellera Reine327e).

★ Note: Une reine n'a pas de reine.

Il ne reste plus qu'à pouvoir créer des fourmis, et ça c'est le boulot de la reine.

12. Créons la méthode replicate

❏ Celle-ci renvoie une Fourmi toute neuve.

★ Attention, créer une fourmi est coûteux. (100 d'énergie).

Reine
replicate()

Il va donc falloir que la reine absorbe quelques choses si besoin.

★ Le nom de chaque nouvelle fourmi sera le nom de sa reine suivi par un numéro de série. Le numéro de série est donc le nombre de fourmi créés par cette reine.

Exemple : Reine327e crée trois fourmis, elles se nommeront Reine327e1,Reine327e2 et Reine327e3.

13. Ajoutons un numéro de série incrémental

❏ Cell-ci nous permettra d'identifier les nouvelles fourmis.

Reine
nb_enfants
replicate()

★ Le numéro est donc propre à chaque reine, ce qui veut dire qu'il ne peut pas être static.

★ Il doit donc être implémenté comme attribut de Reine, et bien s'incrémenter à chaque fois qu'elle crée une fourmi.

Note : la fonction to_string existe, mais suivant l'IDE et le compilateur que vous utilisez, elle peut bugger (sur codeblocks par exemple…) au cas où, voici une réécriture de la fonction, à mettre dans votre .cpp:

```
#include <sstream>
template <typename T>
std::string to_string(T value)
{
	std::ostringstream os ;
	os << value ;
	return os.str() ;
}
```

Enfin, on aimerait connaitre le nombre total de fourmis créées depuis le début.

14. Créons donc un attribut static

❏ Celui-ci sera dans notre classe Fourmi, et nous renseignera sur le nombre total de fourmis.

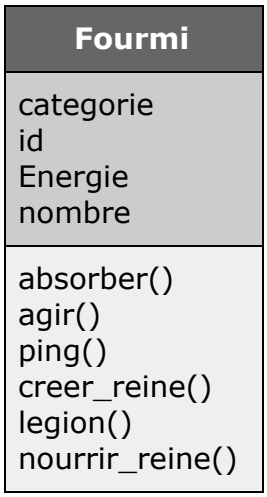
Fourmi
categorie id Energie nombre
absorber() agir() ping() creer_reine() legion()

❏ Cette information peut être obtenue en demandant à n'importe quel fourmi leur nombre (méthode int legion()). Attention aux constructeurs et destructeurs !

La reine est cependant peu mobile, elle a besoin de ses fourmis pour se nourrir.
Il faut donc permettre aux fourmis de pouvoir donner de leur énergie à leur reine.

16. Codons la méthode nourrir_reine()

Cette méthode de chaque fourmi lui permet de transférer une partie de son énergie à sa reine.



Pour les plus avancés, on va maintenant créer un monde (une succession de choses.)

17. Créez un vector de choses

❏ Ce vector sera à remplir de choses. (voir vector, push_back(), pop_back()...)

★ A chaque fois qu'un fourmi va vouloir absorber une chose, on la retire des choses disponibles sur le monde. Attention aux cas où le monde est vide !

Enfin, chaque reine doit pouvoir savoir quels fourmis elle a créé.

18. Ajoutons un attribut vector

❏ Ce vector de pointeurs sur fourmis, appelé cohorte, contiendra l'adresse de toutes les fourmis créées par cette reine.

19. Ajoutons une méthode ping_cohorte()

Cette méthode nous permettra de pinger toutes les fourmis créées par la reine.

Pour aller plus loin :

Nous avons la reine de nos fourmis, ils peuvent consommer et se reproduire. Libre à vous d'implémenter un monde autour de ça.

Idées :

Faire une carte avec un visuel des fourmis. On commence avec une fourmi et beaucoup d'objets. Le fourmi doit aller vers les objets pour les consommer. Faire cela lui coûte de l'énergie. Une fois qu'elle en a accumulé suffisamment, elle peut créer une reine immobile. Celle-ci va utiliser l'énergie que ses fourmis lui apportent pour continuer à produire d'autres fourmis. Ajouter une IA de recherche des objets sur le monde, et observer les fourmis se multiplier et se débarrasser des déchets.

★ PS: Cela constitue une base tout à fait acceptable pour un éventuel projet C++...